Orchestration Server Developer's Guide

Orchestration Server 8.1.4

7/18/2022

Table of Contents

Orchestration Server 8.1.4 Developer Guide

SCXML Language Reference

Orchestration applications are created by writing SCXML documents either via your favorite text editor or via Genesys Composer. SCXML is a W3C evolving standard which allows applications to be described or represented in a state machine execution language.

See SCXML Language Reference.

Orchestration Extensions

Orchestration Extensions are provided in the form of one or more functional modules that package up related actions, objects, functions and events that enable developers to interact with the Genesys platform without having to directly leverage existing SDKs. This allows developers from within Orchestration to create feature rich and open applications.

See Orchestration Extensions.

Migration from IRD

For customers with existing IRD strategies, Composer can be used to help migrate your existing strategies. For information on the various mappings from IRD functions and objects please see IRD To Composer Migration Guide. For supplementary information please refer to Migration from IRD.

Orchestration Integration

To facilitate richer application development Orchestration enables other enterprise applications and systems within your environment to interact with it via open standard web based interfaces. By enabling this it allows the full potential of Orchestration within an enterprise setting to be realized and allows developers to build applications that extended past the capabilities of what is natively supported by Orchestration. To allow other applications within your enterprise to directly interact with Orchestration and the sessions that it is executing we support an external interface which is described here in External Interfaces and to enable Orchestration sessions to reach out and interact with other systems we support our Orchestration Integration Interfaces which is described here Orchestration Server Integration.

Getting Started Guide

This guide is intended as a reference for those working with Orchestration for the first time, but can also serve as a reference for common use cases. For first time users, it is recommended you walk through the exercises sequentially as each section builds on the previous section. For users that is familiar with Orchestration, this guide will provide various samples for common tasks.

See Orchestration Getting Started Guide.

How-To

This topic provides details on how to perform common tasks using SCXML and/or Orchestration

See Orchestration Server How-To.

Samples and Templates

This collection of SCXML files will help you develop your Orchestration Server applications. It gives examples of typical use cases which serve as building blocks for your SCXML applications.

See Samples and Templates.

Troubleshooting

See Orchestration Server Troubleshooting.

Document Change History

The following topics are new or have been updated in the *Orchestration Developer's Guide*:

Orchestration Server Release 8.1.400.12

New “urs” method is added to the `session:fetch` Action Element. It simplifies usage of Universal Routing Server functions, previously available for Orchestration Server only via the Universal Routing Server Web interface. See `<fetch>` under the Action Elements section.

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!"
```

Contents of SCXML Language Reference

The organization and content of the SCXML Language Reference is detailed below. Numbers are included on this page so you can see the items that fall under every subtopic.

Orchestration Extensions

_genesys Object

Every SCXML session will have a global root object (_genesys) from which an application will have access to objects, properties, and functions to access Orchestration extensions. Each used extension is accessed as a property of _genesys. For example, the Interaction Interface and the Queue Interface will have the corresponding root object off of the _genesys object: _genesys.ixn and _genesys.queue.

Extension Namespaces

Each of the Orchestration Extensions have an associated set of namespaces for actions, events, content, and functions. The following are the namespaces for the Orchestration Extensions:

Extension	Action Namespace*	Event prefix	Object and Function root
Core Extension - Session	*www.genesyslab.com/ modules/session*	session.xxx	_genesys.session.xxx
Core Extension - Web	*www.genesyslab.com/ modules/ws*	ws.xxx	_genesys.ws.xxx
Queue	*www.genesyslab.com/ modules/queue*	queue.xxx	_genesys.queue.xxx
Interaction	*www.genesyslab.com/ modules/interaction*	interaction.xxx, voice.xxx, msgbased.xxx, chat.xxx	_genesys.ixn.xxx
Dialog	*www.genesyslab.com/ modules/dialog*	dialog.xxx	_genesys.dialog.xxx
Statistics	*www.genesyslab.com/ modules/statistic*	statistic.xxx	_genesys.statistic.xxx
Classification	*www.genesyslab.com/ modules/classification*	classification.xxx	_genesys.classification.xxx
Resource	*www.genesyslab.com/ modules/resource*	resource.xxx	_genesys.resource.xxx
Elasticconnector	*www.genesyslab.com/ modules/ elasticconnector*	elasticconnector.xxx	_genesys.elasticconnector.xxx
Agent	*www.genesyslab.com/ modules/agent*	agent.xxx	_genesys.agent.xxx

***** The namespaces used by an application must be included in the `<scxml>` element using the `xmlns` attribute.

Core Extensions (Session and Web)

These interfaces provide functionality that is not specific to an external interface. This includes capabilities for creating sessions and accessing external http based servers.

See Core Extensions.

Queue Interface

This Interface provides the ability to allow the orchestration logic to request the appropriate resource for some processing and return the appropriate address information for the resource. The current URS functionality (queuing, prioritization, and so on) will be used for this interface. In addition to core queuing and target selection, this functional module interface will also support outbound interaction processing.

See Queue Interface.

Interaction Interface

An interaction represents the various types of communications between a resource and a customer:

- Conversation-based communication between a customer and given contact center or enterprise resources using a single logic media (certain media may support more than one media type over an interaction (voice and video) (for example, a voice call or a chat session). This type of interaction involves a series of information being communicated back and forth.
- Message-based communication with a customer (for example, an incoming e-mail or sms or an outgoing e-mail or sms).

See Interaction Interface.

Dialog (Treatment) Interface

The dialog interface defines the functionality that allows Orchestration logic to do the following:

- Run a particular dialog application (VXML, HTML, and so on) on a specific interaction and by a specific resource.
- Collect the results of the dialog application that was run.

See Dialog Interface.

Statistic Interface

This interface provides statistical information to the orchestration logic.

See Statistic Interface.

Classification Interface

This interface provides the ability to classify and screen interaction content to help the orchestration logic determine what the customer wants.

See Classification Interface.

Resource Interface

This functional module contains enumeration objects that can be used in other functional modules.

See Resource Interface.

Elasticconnector

This extension helps remove the management of connectivity to Elasticsearch cluster from the SCXML strategy. It also provides simplified access to Elasticsearch APIs.

See Elasticsearch Connector.

Agent

This extension is used for implementing agent management features like logout, DND, and ability to change agent state for voice media.

See Agent Extension.

Queue Interface

This Interface provides the ability to allow the orchestration logic to request the appropriate resource for some processing and return the appropriate address information for the resource. The current URS functionality (queuing, prioritization, and so on) will be used for this interface.

Target Formats

All target definitions are strings, but the strings have a particular format, depending on where they are used.

The following is the list of target formats that are used:

- **Target DN:** `<number>@<switch>`.DN. This is the format that is used for a DN that is being used as a target. The following are the details of the different elements of the sub-strings:
  - number is the address of the target resource.
  - switch is the switch or media server that controls this address.
  - DN is the type of target.
- **Possible Target:** `{threshold}[weight] name@server.type` or `{threshold}[weight]?name:skillexpression@server.type`. This is the format that is used when defining a target that should be considered as a possible target or group of targets during the target selection process started by the `<submit>` action. The following are the details of the different elements of the sub-strings.
  - weight - same as defined in `<target>` element
  - name - same as defined in `<target>` element
  - server - same as defined in statserver attribute in the target> element
  - type - same as defined in `<target>` element but with the following abbreviations:
    - A (for agent),
    - AP (for agent place),
    - GA (for group of agents),
    - GP (for group of places),
    - Q (for Queues (real and virtual)),
    - DN (for DN),
    - GA (for skill),
    - C (for campaigns),
    - GC (for group of campaigns),
    - RP (for routing points (real and virtual)),

        - DL (for destination label),
        - GQ (for group of queues (DNs)).
    - skillexpr - same as defined in `<target>` element
    - threshold - same as defined in `<target>` element

Skill Expressions

All skill expression definitions are strings, but the strings have a particular format. A skill expression is a set of conditional expressions that are linked together via logic operators. When a skill expression contains more than one conditional expression, the conditional expression to the left has precedence over the conditional expression to the right. Parentheses can be used to overrule this precedence. Your can use an ECMAScript logic expression to create the skill expression string. The following are the parts of the skill expression:

- **Conditional Expression** - This is an expression that produces a result of true or false. This expression has the following format and elements: **dataname operator value** - for example; english >3
    - **dataname** - This element is a string which represents the name of these data types:
        - **Skill** - This is the name of a skill defined in the configuration layer. Skill names are limited to alphanumeric characters and underscores. The name cannot begin with a digit. The name of a skill cannot exceed 126 characters in length.
        - **Statistic** - This is the name of a statistic defined in the configuration layer. The statistic name in a skill expression can be any agent statistic used in the function SData. It must be written in the format: $(statisticname). For example; $(StatAgentLoggedIn)=1
    - **operator** - This is the operator to be used to evaluate the condition. The following are the operators that are supported:
        - != - The meaning or support is different depending on the data type:
            - Skill - not equal to the indicated level value
            - Statistic - not equal to the indicated statistic value
        - < (<=;) - The meaning or support is different depending on the data type:
            - Skill - less than the indicated level value. **Note:** depending on how you use this operator, it may result in including agents that do not have the skill at all (skillname = 0). For example, with English <= 8, the queue functional module includes all agents with the English skill less than 8, and also agents with no English skill at all.
            - Statistic - less than the indicated statistic value
        - <= (<=) - The meaning or support is different depending on the data type:
            - Skill - less than or equal to the indicated level value
            - Statistic - less than or equal to the indicated statistic value
        - = - The meaning or support is different depending on the data type:
            - Skill - equal to the indicated level value
            - Statistic - equal to the indicated statistic value

- > (>) - The meaning or support is different depending on the data type:
  - Skill - greater than the indicated level value
  - Statistic - greater than the indicated statistic value
- >= (>=) - The meaning or support is different depending on the data type:
  - Skill - greater than or equal to the indicated level value
  - Statistic - greater than or equal to the indicated statistic value

- **value** - This is a value of the same data type as the dataname element. The value must already evaluate to an integer. Floating point numbers are not supported. There are different limitations depending on the data type:
  - **Skill value** - This value represents the level of skill - for example, is the agent's English skill level greater than 3 `(English > 3)`. An agent can be excluded from a skill by setting that agent's skill level for that skill to zero in the configuration layer (that is, English =0).
  - **Statistic** - This value represents the value of the statistic - for example, has the agent been ready longer than 20 seconds`($(StatTimeInReadyState) > 20)`.

- **Logic Operators:** The logic operators are a way to evaluate multiple conditional expressions together. The following logic operators are supported: AND(&- &) and OR (|). The AND and OR logic operators have the same priority. For example; "`English > 3 & $(StatAgentLoggedIn)=1`"

A skill expression has the following limitations:

- The maximum size of an overall skill expression (as text) is 10239 bytes.
- A skill expression should have no more than 25 constructions, such as English > 1.

Routing Rules

Currently, Universal Routing Server supports the use of routing rules which are stored in Configuration Server. These are really not rules, but routing profiles which define a unique set of attributes and child elements for a `<submit>` element. An application can use the `<submit>` src attribute for using routing rules. These routing rule definitions can come from transaction objects from Configuration Server.

For details, see <submit>. The following are the properties for each routing rule and how they map to the `<submit>` attributes and child elements:

Routing Rule	Routing Rule Property	Queue Functional Module Interface equivalent
LoadBalancing	VQ name	`<submit>` queue attribute
	List of ACDQueue data (switch and name)	`<targets>` and `<target>` or `<targetid>`
Percentage		`<submit>` ordertype attribute

Routing Rule	Routing Rule Property	Queue Functional Module Interface equivalent
		="percentage"
	VQ name	`<submit>` queue attribute
	Type of targets	`<targets>` type attribute
	List of targets and percentages	`<targets>` `<target>` (name and weight attributes)
	Busy treatment data	`<runtreatments>` and all child elements
Service Level		`<submit> ordertype` attribute = "`min`" or "`max`"
	VQ name	`<submit> queue` attribute
	Statserver name	`<targets>` `<target>` `statserver` attribute
	Statistic name	`<submit> orderstat` attribute
	Skills to use and their skill levels	`<target> skillexpr` attribute
	Service Factor data (distribute X percent of interaction in Y seconds)	Not supported through an attribute or child element but can be supported through `<submit> src` attribute and the `gdata` scheme.
	Busy treatment data	`<runtreatments>` and all child elements
	Importance	Not supported through an attribute or child element but can be supported through `<submit> src` attribute and the `gdata` scheme.
Statistics		`<submit>` ordertype attribute = "min" or "max"
	VQ name	`<submit>` queue attribute
	Statserver name	`<targets>` `<target>` statserver attribute

Routing Rule	Routing Rule Property	Queue Functional Module Interface equivalent
	Statistic name	`<submit>` orderstat attribute
	Type of targets	`<targets>` type attribute
	List of target names	`<targets>` `<target>` or `<targetid>`
	Busy treatment data	`<runtreatments>` and all child elements
Workforce	VQ name	`<submit>` queue attribute
	Schedule data (activity name, cut off time)	`<activity>`
	Busy treatment data	`<runtreatments>` and all child elements

Object Model

_genesys.queue Object

This is the global root object for the Queue functional module interface. This object is maintained by the Queue functional module that implements this interface.

The name of the object will be "**_genesys.queue**".

There are currently no data properties associated with this object.

_genesys.queue.overwriteType ENUM Object

This represents the DNIS overwrite type enumeration. This enumeration is maintained by the orchestration platform.

This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
UseNone	read only	integer	none	0	Use nothing to overwrite the DNIS.

Name	Access	Type	Default Value	Valid Values	Description
UseANI	read only	integer	none	1	Use the ANI value to overwrite the DNIS.
UseDNIS	read only	integer	none	2	Use the DNIS value to overwrite the DNIS.
UseConfig	read only	integer	none	3	Use a configuration value to overwrite the DNIS.
UseValue	read only	integer	none	4	Use the supplied value to overwrite the DNIS.

_genesys.queue.rType ENUM Object

This represents the rType enumeration. This enumeration is maintained by the orchestration platform.

This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
RouteTypeUnknown	read only	integer	none	0	This represents an unknown route type
RouteTypeDefault	read only	integer	none	1	This represents a default route type
RouteTypeLabel	read only	integer	none	2	This represents a label route type
RouteTypeOverwriteDNIS	read only	integer	none	3	This represents an overwrite DNIS route type
RouteTypeDDD	read only	integer	none	4	This represents a DDD route type
RouteTypeIDDD	read only	integer	none	5	This represents an IDDD route type

Name	Access	Type	Default Value	Valid Values	Description
RouteTypeDirect	read only	integer	none	6	This represents a direct route type
RouteTypeReject	read only	integer	none	7	This represents a reject route type
RouteTypeAnnouncement	read only	integer	none	8	This represents an announcement route type
RouteTypePostFeature	read only	integer	none	9	This represents a post feature route type
RouteTypeDirectAgent	read only	integer	none	10	This represents a direct agent route type
RouteTypePriority	read only	integer	none	11	This represents a priority route type
RouteTypeDirectPriority	read only	integer	none	12	This represents a direct priority route type
RouteTypeGetFromDN	read only	integer	none	13	This represents a from DN route type
RouteTypeAgentID	read only	integer	none	14	This represents an agent ID route type
RouteTypeCallDisconnect	read only	integer	none	15	This represents a call disconnect route type

_genesys.queue.quotaType ENUM Object

This represents the quotaType enumeration. This enumeration is maintained by the orchestration platform.

This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
QuotaMin	read only	integer	none	0	This means to filter

Name	Access	Type	Default Value	Valid Values	Description
					on the minimum quota values
QuotaTarget	read only	integer	none	1	This represents means to filter on the target-based quota values
QuotaMax	read only	integer	none	2	This means to filter on the maximum quota values

_genesys.queue.statcond ENUM Object

This represents the statcond enumeration. This enumeration is maintained by the orchestration platform.

This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
ReadyIfLess	read only	integer	none	0	This represents the condition where the agent is ready and the associated statistic is less than the threshold value.
ReadyIfGreater	read only	integer	none	1	This represents the condition where the agent is ready and the associated statistic is greater than the threshold value.
ReadyIfNotGreater	read only	integer	none	2	This represents the condition where the agent is ready and the associated statistic is not greater than the threshold value.
ReadyIfNotLess	read only	integer	none	3	This represents the condition where the agent is ready and the associated statistic is not less than the threshold

Name	Access	Type	Default Value	Valid Values	Description
					value.

_genesys.queue.usecapcond ENUM Object

This represents the usecapcond enumeration. This enumeration is maintained by the orchestration platform.

This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
OnStatError	read only	integer	none	0	This indicates to use the statistical tables from the configuration layer when there is an error with the Stat Server results.
Never	read only	integer	none	1	This indicates to never use the statistical tables
Only	read only	integer	none	2	This indicates to only use the statistical tables.

Functions

_genesys.queue.reserveTarget

When the `<submit>` action returns a ready target, the target is always blocked by the functional module during some time in seconds (the value of the platform's *transition_time* configuration option). The reserveTarget function allows you to override such behavior and change the time interval or even cancel it entirely (*time* parameter = 0). During the time the target is reserved, the functional module does not distribute any interactions to the target as specified by the target parameter, which is the result of the `<submit>` action. This function can be used in cases where a target may have additional conditions (except a Stat Server-reported not-ready state) that should prevent the target from being selected as a valid target. This blocking time will be applied to all `<submit>` action resulting targets until it is changed again by this function.

`void _genesys.queue.reserveTarget(ixnid, target, time)` Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.

- **target:** STRING which can be a variable or a constant - This is the name of the target to be reserved in the target DN format.
- **time:** INTEGER which can be a variable or a constant - This is the time interval in seconds to reserve this target for (exclude from target selection process). A value of zero (0) will cancel it entirely.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.reserveTarget( _data.ixnid, 'return:ok|n:_data.reqid|agent:703_sip|place:703|
dn:703|switch:SipSwitch', 40 );
```

_genesys.queue.cCTExtractTargets

This function produces a list of targets in "possible target" format from the supplied arguments and stores target DN information associated to each target for subsequent use in number translation of type`[TARGET.CCTN].`These are maintained for the life of the session by the functional module and can be used for any interaction associated with session.

This function supports these target types:

- Agent (A)
- Place (P)
- Agent Group (GA)
- Place Group (GP)

`targets _genesys.queue.cCTExtractTargets(ixnid, statserver, input)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **statserver:** STRING which can be a variable or a constant - This parameter is used as the location attribute for each of the targets listed in the output.
- **input:** STRING which can be a variable or a constant - This parameter must have the following key/value list of the form: TargetID1.TargetType1:Value1|...|TargetIDN.TargetTypeN:ValueN with no spaces in the list.

Returns: `targets`: STRING-The result of the function is a string of the form: `"TargetID1@StatServer.TargetType1,...,TargetIDN@StatServer.TargetTypeN"`. Thus, the targets listed in the result are separated by commas and contain no spaces.

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.cCTExtractTargets( _data.ixnid, 'StatServer', '701.A|702.P|
SipGr_3.GA|SipPlGr2.GP' );
```

_data.myResult will be the following string:

```
'701@StatServer.A,702@StatServer.P,SipGr_3@StatServer.GA,SipPlGr2@StatServer.GP'
```

_genesys.queue.checkAgentState

This function instructs the functional module for the associated session (across all associated interactions) as to whether to take into account the state of an Agent, Place, Agent Group, or Place Group as reported by Stat Server or to look only for free DNs belonging to the Agent, Place, or Agent Group. For example, `_genesys.queue.checkAgentState(false)` makes it possible to route a voice interaction to an agent that Stat Server reports as not ready. If agent capacity rules are set, this function has no effect (as the Genesys Agent Capacity model does not use agent state).

To allow an agent to receive multiple voice interactions, this function must be `false`. However, the side effect is that the functional module distributes interactions so as to occupy all DNs of Agent A before considering Agent B. When this function is set to `false`, or the function `_genesys.queue.useAgentState` is used, the functional module does not apply the *verification_time* configuration option to agents, but still applies it to agent DNs.

`void _genesys.queue.checkAgentState(ixnid, check)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **check:** BOOLEAN which can be a variable or a constant - This parameter identifies whether the functional module should use the agent state from StatServer or not.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.checkAgentState( _data.ixnid, false);
```

_genesys.queue.clearThresholds

This function invalidates all thresholds previously set by `<submit>` actions for a given interaction. As a result, the functional module now considers all targets that were previously affected by `<submit>` actions as unconditionally ready for routing.

`void _genesys.queue.clearThresholds(ixnid)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.clearThresholds( _data.ixnid );
```

_genesys.queue.countSkillInGroup

This function determines the number of agents with a skill set or statistical parameters that satisfy the indicated skill expression. It returns the number of the agents belonging to the agent group based on the defined skill expression. You can also just specify a Stat Server and a skill expression without specifying an Agent Group. However, a skill expression and a Stat Server must be specified. The Stat Server is used to query the content of the provided Agent Group (real or virtual).

`total _genesys.queue.countSkillInGroup(ixnid, statserver, group, sexpr)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **statserver:** STRING which can be a variable or a constant - This parameter is optional. This parameter is the name of the Stat Server containing information on the agents for this function. If not supplied, the Stat Server used will be the one defined by the default_stat_server configuration option of the platform.
- **group:** STRING which can be a variable or a constant - This parameter is the agent group for the Stat Server that this function checks against. It can either be an Agent Group name or a list of comma-separated list of Agents, Agent Groups, Places, or Place Groups. Agents are included in this group either by placing the agent name from the Persons folder into the Agent Groups folder or defining a virtual group using a skill expression within the Annex tab of the Agent Group object.
- **sexpr:** STRING which can be a variable or a constant - This parameter defines the skill expression string used to evaluate the agents. It can use skills, variables, numeric constants, and statistics to filter out agents based on their state. The statistic name in a skill expression can be any agent statistic used in the function SData. It must be written in the format:`$(statistic)`. This ability to use statistics in a skill expression allows you to conduct queries based on a statistic. For example, if you want to query the number of agents with a Spanish skill of at least 5 who are logged in, the expression would be as follows: `'Spanish >= 5 & $(StatAgentsLoggedIn)=1'.` This can be passed to the function as a literal string or as an an ECMAScript expression such as '`Spanish >=' + level + ' $(StatAgentsLoggedIn)=' + minAgents` where level and miNAgents are variables defined else where in your applciation.

Returns: `total`: NUMBER - The result of the function is an integer which represents the number of agents that meet the critieria of the skill expression.

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.countSkillInGroup( _data.ixnid, 'StatServer', 'SipGr_2',
'english > 3 & french > 4' );
```

_data.myResult will be the following:

```
5
```

_genesys.queue.createSkillGroup

This function converts the provided Agent Group, Skill Expression, and StatServer into a normal target format that represents all agents belonging to the Agent Group supplied as a parameter that satisfies the logical condition given by the Skill Expression.

`tlist _genesys.queue.createSkillGroup(ixnid, statserver, group, sexpr)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **statserver:** STRING which can be a variable or a constant - This parameter is optional. It contains the name of the Stat Server containing information on the agents for this function. If not supplied, the Stat Server used will be the one defined by the *default_stat_server* configuration option of the platform.
- **group:** STRING which can be a variable or a constant - This parameter is the agent group for the Stat Server that this function checks against.
- **sexpr:** STRING which can be a variable or a constant - This parameter defines the skill expression used to evaluate the agents. This parameter can use skills, variables, numeric constants, and statistics to filter out agents based on their state. The statistic name in a skill expression can be any agent statistic used in the function SData. It must be written in the format: `$(statistic)`. This ability to use statistics in a skill expression allows you to conduct queries based on a statistic. For example, if you want to query the number of agents with a Spanish skill of at least 5 who are logged in, the expression would be as follows: `Spanish >= 5 & $(StatAgentsLoggedIn)=1`.

Returns: `tlist`: STRING - The result of the function is a string which represents a list of targets (each target is comma-separated in the string) that meet the critieria of the skill expression. It is of the form:`"?GroupName:SkillExpression@statserver.GA".`

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.createSkillGroup( _data.ixnid, 'StatServer', 'SipGr_2',
'english>3' );
```

_data.myResult will be the following string:

```
'?SipGr_2:english>3@StatServer.GA'
```

_genesys.queue.excludeAgents

This function instructs the functional module not to route interactions to any agent on the specified list of agents. This applies across all `<submit>` actions associated with the session.

`prev_agents _genesys.queue.excludeAgents(ixnid, agents)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **agents:** STRING which can be a variable or a constant - This parameter is a comma-separated list of agent IDs.

Returns: `prev_agents`: STRING - The result of the function is a string which represents a list of previous excluded agents. Each agent ID is comma-separated in the string.

The following is an example of input and output for this function:

```
_data.myResult1 = _genesys.queue.excludeAgents( _data.ixnid, '702_sip' );
...
_data.myResult2 = _genesys.queue.excludeAgents( _data.ixnid, '703_sip' );
```

_data.myResult2 will be the following string:

```
'"702_sip"'
```

_genesys.queue.extRouterError

This function is used to change the default external routing in the case of a failure to get a remote access number. This is only applicable across all interactions associated with the session when the application is using the `<submit>` action with the route attribute set to true.

`void _genesys.queue.extRouterError(ixnid, enable)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **enable:** BOOLEAN which can be a variable or a constant - If this parameter is set to true, then the functional module handles an external routing failure as a routing error (according to the functional module on_route_error configuration option settings. This prevents the functional module from ignoring any error messages in response to external routing requests and stops the functional module from continuing to attempt to route based on the original remote access number. If set to false, the functional module will continue the attempt to route the call based on the original number. By default, enable is set to false.

Returns: `void`

The following is an example of input and output for this function:

```
_genesys.queue.extRouterError( _data.ixnid, true );
```

_genesys.queue.extRouteStatus

This function returns true if external routing is possible and false if not. `status _genesys.queue.extRouteStatus(ixnid, switch)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **switch:** STRING which can be a variable or a constant - This parameter is a remote location where the interaction is being routed (that is, the switch ID).

Returns: `status`: BOOLEAN - The result of the function is a boolean which indicates whether external routing is possible for the given media server (switch).

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.extRouteStatus( _data.ixnid, 'SipSwitch' );
```

_data.myResult will be the following string:

```
true
```

_genesys.queue.findServiceObjective

This function returns a value that is defined in a configuration layer objective table object for a given a Service Objective, which is a unique combination of Customer Segment, Service Type, and Media Type. If the *Update* parameter is true, the Service Objective with Service Type and Customer Segment are automatically attached to the interaction.

`value _genesys.queue.findServiceObjective(ixnid, table, media, service, segment, update)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **table:** STRING which can be a variable or a constant - This parameter is the name of the objective table in the configuration.
- **media:** NUMBER which can be a variable or a constant - This parameter is the desired media type in integer form.
- **service:** STRING which can be a variable or a constant - This parameter defines the type of service desired.
- **segment:** STRING which can be a variable or a constant - This parameter defines the type of customer segment desired.
- **update:** BOOLEAN which can be a variable or a constant - This parameter defines whether the interaction should be updated with a value as well as the properties of the service objective (media type, service type, and customer segment).

Returns: `value`: NUMBER - The result of the function is an integer which represents the configured value for this service objective. Zero is returned if there are any issues (unsuccessful search - no service objective defined) while processing this function.

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.findServiceObjective( _data.ixnid, 'OBT3', 0, 'help',
'Bronze', true );
```

_data.myResult will be the following number:

```
150
```

_genesys.queue.incrementPriority

This function results in incrementing the selected interaction priority by the `increment` every `interval` second. It affects the priority of the interaction for `<submit>` actions the interaction is already waiting for and those it may be waiting for in the future.

`void _genesys.queue.incrementPriority(ixnid, increment, interval)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have its priority incremented.

- **increment:** NUMBER (INTEGER) which can be a variable or a constant - This parameter is the increment by which the priority is to be adjusted.
- **interval:** NUMBER (INTEGER) which can be a variable or a constant - This parameter is the time interval at which the priority is to be incremented. **Note:** the interval cannot be less than 5 seconds.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.incrementPriority( _data.ixnid, 2, 10 );
```

_genesys.queue.priorityLimits (since ORS 8.1.200.40 and URS 8.1.200.21)

This function results in imposing additional limits on values of priority the interaction can have in any routing queue. Without it interaction priority can accept any value in range [-1,000,000,000 : 1,000,000,000].

`void _genesys.queue.priorityLimits(ixnid, min, max)`

Parameters (all parameters are mandatory):

- **ixnid:** STRING which can be a variable or a constant - This parameter defines the ID of the interaction to which provided priority limits will be applied.
- **min:** NUMBER (INTEGER) which can be a variable or a constant - This parameter is the minimal possible value for the interaction priority in any routing queue.
- **max:** NUMBER (INTEGER) which can be a variable or a constant - This parameter is the maximal possible value for the interaction priority in any routing queue.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.priorityLimits( _data.ixnid, 1, 10 );
```

_genesys.queue.nMTExtractTargets

This function enables you to track the number of active interactions at a device that is not configured in the Configuration Database. It establishes a counter for all active interactions at the non-configured device. Using this counter, you can compare the number of active interactions to a specified threshold and stop routing interactions to the device when the threshold has been reached.

This function parses a list of targets, which is a comma-separated list of attributes produced by a database query or some outside application (an application on an application server, for example) such as `<fetch>`. Each target can be described using four attributes: *Target Name, Threshold, Speaking Time*, and *RetryTime*. These attributes are explained below:

- **Target Name** - This identifies the target.
- **Threshold** - The maximum number of interactions at the target when this target is considered to be ready. The threshold value is considered as attribute to an interaction. This function sets this attribute for each interaction it evaluates. When it is time for the interaction to be routed to the non-configured device, the functional module compares the interaction's threshold attribute to the counter. If the

counter shows a value greater than the interaction's threshold attribute, the interaction is not routed until the counter's value drops below that threshold.

- **Speaking Time** - The functional module considers an interaction to a non-configured device to be terminated when the controlling server (for example, T-Server) reports that it has been terminated. However, if no information is received from the controlling server (for example, T-Server) in the time specified by the Speaking Time (in seconds), the functional module considers the call to be terminated.
- **Retry Time** - If the functional module receives an error message in response to a routing request to a non-configured device that indicates that the number of interactions at the target is different from the number the functional module believes are there, the functional module temporarily stops sending interactions to that device for the amount of time specified by this element (Retry Time). This delay enables synchronization of the number of interactions on the device with the number that the functional module believes are there.

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.nMTExtractTargets( _data.ixnid, '701,702,703', 1, 2, 5, 3 );
```

_data.myResult will be the following string:

```
'{RStatCallsInQueue<=2}701.DN,{RStatCallsInQueue<=2}702.DN,{RStatCallsInQueue<=2}703.DN'
```

Using the Function

Use of this function to track the number of interactions at non-configured devices usually includes:

- Setting a threshold, using the `_genesys.queue.setThresholdEx` function, for every nonconfigured device that you want to include. The only threshold you can set for non-configured devices applies to the value returned by the `RStatCallsInQueue` statistic. Alternatively, you can set a threshold by prefixing `Threshold{Statistic op value}` before the target specification. When you set a threshold in this way, this function automatically augments the targets with prefixes in the indicated format. The output of this function is a comma-separated list of targets, with thresholds, that is ready to be used as a parameter of standard target-selecting functions or objects.
- Configuring the functional module to count the `RStatCallsInQueue` statistic for nonconfigured devices.
- Setting all the non-configured devices as targets in `<submit>` actions.

Note: The functional module has no information about the current state of the device - only the number of interactions active on it. Active interactions are considered to be the number of interactions the functional module has sent to the target, minus the number of interactions that were terminated. In addition to generating output, this function also overwrites the default values (15 seconds and 600 seconds) for *Retry Time* and *Speaking Time*. Thresholds for non-configured devices set using the `_genesys.FMname.SetThresholdEx` function, for example, use the current default values for *Retry Time* and *Speaking Time* that were set by the latest invocation of this function.

Note: Changes applied to default settings (*RetryTime, Speaking Time*) specified in this function work only for non-configured devices that were created after this function was used. Those non-configured devices that were created previously are not affected by any changes made using this function. If the information about the non-configured devices (numbers, threshold values) is retrieved from an outside source (for example, a database), use this function to transform the result of this query into a target list suitable for use by the `<submit>` action. In this case, the use of this function automatically sets all the specified thresholds and you do not need to call the `_genesys.FMname.setThresholdEx` function.

Setting Multiple Thresholds

You can set more than one threshold by using this function in more than one logic definition or you can use it multiple times in the same logic. In this way, you can set a different allowable number of concurrent interactions for different situations, as described below.

Note: If you use the same non-configured device in other logic definitions or in multiple places in the same logic, the functional module uses the same counter for that device. So this definition is global across the functional module.

Using `NMTExtractTargets` to Set Multiple Thresholds - Example:

You can set different thresholds for the same non-configured device to take account of different business conditions. For example, between 9:00 AM and 5:00 PM, an outsourcer with a nonconfigured device can handle 100 concurrent interactions. After 5:00PM, the outsourcer can handle only 50 concurrent interactions on this device.

Here is how it works:

1. Place this function in two places. You can use it twice in one logic definition or use two separate logic definitions to handle this situation, depending on what is best suited to your environment. A global counter is created for the non-configured device.
2. Using one instance of this function, set a default threshold value of 100 to be attached as an attribute to all interactions that arrive between 9:00 AM and 5:00 PM.
3. Using the other instance of the function, set a default threshold value of 50 to be attached as an attribute to all interactions that arrive between 5:00 PM and 9:00 AM.

When an interaction is ready to be routed, the interaction attribute is compared to the global counter, which is incremented or decremented to correspond to the number of active interactions on this non-configured device. If the value on the counter is greater than the interaction's Threshold attribute, the interaction is not routed until the counter's value drops below the appropriate threshold.

Deployment Considerations

Only one orchestration platform can send interactions to the non-monitored device. If you require multiple numbers to distribute to the same non-monitored device, align the SN-to-DN table, the network switch, and platform so that all interactions sent to a nonmonitored device are processed by the same platform. You can use a network-controlling server (for example, T-Server or a SIP Server) as the network switch. To use this function, the platform and functional module must register on a special device (called switch::) in order to receive notification from the controlling server (for example, T-Server) about the termination of interactions. If you do not register, the *RStatCallsInQueue*statistic is not correctly calculated for non-monitored targets. Registration is not automatic and is controlled by platform (using the *call_monitoring* configuration option).

```
tlist _genesys.queue.nMTExtractTargets(ixnid, targets, dThreshold, dSpeakTime,
dRetryTime, recordSize)
```

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory It defines the ID of the interaction which should have its priority incremented.
- **targets:** STRING which can be a variable or a constant - This parameter is the list of comma-separated

targets and their attributes. The recordSize parameter will identify how attributes are associated with a target definition.

- **dThreshold:** NUMBER (INTEGER) which can be a variable or a constant - This parameter is the default maximum number of interactions at the target when this target is considered to be ready. This parameter will be used for targets that do not have this attribute specified in the targets parameter.
- **dSpeakTime:** NUMBER (INTEGER) which can be a variable or a constant - This parameter is the default maximum amount of speak time for an interaction. If the functional module does not receive an event indicating that the interaction has terminated with in this time frame, the functional module will consider it terminated. This parameter will be used for targets that do not have this attribute specified in the targets parameter.
- **dRetryTime:** NUMBER (INTEGER) which can be a variable or a constant - This parameter is the default maximum amount of retry time for an interaction. If the functional module failed to route the interaction, it will wait for this amount of time before trying to route the interaction again. This parameter will be used for targets that do not have this attribute specified in the targets parameter.
- **recordSize:** NUMBER (INTEGER) which can be a variable or a constant - This parameter defines how many attributes are entered to describe each target. If some of the four elements listed in the following bullets [I don't see the bullets...] are absent (only Target name is mandatory) the corresponding default parameter values are used instead.

Returns: `tlist`: STRING - The result of the function is a string of the form: "{RStatCallsInQueue<=TargetThres1}TargetName1.DN,...,{RStatCallsInQueue<=TargetThresN}TargetNameN.DN". Thus, the targets listed in the result are separated by commas and contain no spaces.

_genesys.queue.onRouteError

This function allows you to specify an individual functional module reaction for every type of error. If used, this option overwrites the*on_route_error* configuration option for all interactions associated with the current session. In the case of an error, the functional module behaves in the following way:

- Checks if this function was executed for this type of error. If yes, then the functional module behaves as the function specifies.
- Otherwise, checks if there is a platform-defined default reaction (that is, the configuration option *on_route_error*) for this error. If yes, then the functional module executes this reaction.

Note: This function should ONLY be used when using the `<submit>` action with the route attribute set to true. `void _genesys.queue.onRouteError(ixnid, type, option)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory It defines the ID of the interaction which should have its priority incremented.
- **type:** NUMBER (INTEGER) which can be a variable or a constant - This parameter indicates the type of error that can be encountered while trying to route an interaction.
- **option:** STRING which can be a variable or a constant - This parameter indicates the type of processing the functional module should perform when this type of error is encountered. These options only work for interactions routed to destinations specified in the `<submit>` actions with the route attribute set to true. The values are defined as follows:
  - **try_other** - The Queue functional module will resume waiting for a ready target and try to select another available target. Unlike the reroute value, for which the Queue functional module immediately attempts to reroute the interaction, for try_other the Queue functional module waits

for another target, if none are available before routing the interaction. **Note:** Be sure that the *transition_time* platform configuration option has a value of 3 or higher to enable the Queue functional module to handle the *try_other* setting correctly.

- **strategy_error** - The Queue functional module will abort its current waiting for a ready target, go to error handling, and send the session an error event (`error.queue.submit` with the appropriate error code).

Returns: `void`

The following is an example of input and output for this function:

```
_genesys.queue.onRouteError( _data.ixnid, 140, 'strategy_error' );
```

_genesys.queue.priorityTuning

The functional module always puts interactions into waiting queues according to their priorities. This function defines how the functional module handles interactions with the same priorities. By default, interactions with the same priority are ordered according to the time the interaction began to wait for some target. This applies across all `<submit>` actions.

`void _genesys.queue.priorityTuning(ixnid, useAge,usePredict, useObjective)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory It defines the ID of the interaction which should have its priority incremented.
- **useAge:** BOOLEAN which can be a variable or a constant - This parameter defines whether the age of the interaction should be used to prioritize interactions with the same priority. If this parameter is true, the functional module uses the time the interaction was created instead of the time the interaction is placed into the waiting queue. The age of the interaction is usually the time that the associated session is started for the interaction. Setting the interaction age enables you to safely use multiple `<submit>` actions. The interaction does not lose its position in the queue, because its position is based on the age-of-interaction value, which is not affected by the `<submit>` actions. The following must be done in your logic to use this function.
    - Use function `_genesys.queue.setInteractionAge()` to timestamp the interaction. The age of interaction will be counted from this moment on.
    - Use the function to prioritize this interaction among others in the queue according to their age.
- **usePredict:** BOOLEAN which can be a variable or a constant - This parameter defines if the estimated time for the interaction to be answered will be used instead of the time the interaction has waited when prioritizing interactions with the same priority. If this parameter is true, then the functional module calculates the estimated time for the interaction to be answered and will use this time instead of the time that the interaction has already waited.
- **useObjective:** BOOLEAN which can be a variable or a constant - This parameter defines whether an objective defined in the interaction should be used when prioritizing interactions with the same priority. If this parameter is true, then the functional module calculates the objective defined in the interaction to determine the priority of the interaction.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.priorityTuning( _data.ixnid, true, true, true );
```

_genesys.queue.resetAdjustment

This function cancels any adjustment that may have been set for a statistic for a given target using `_genesys.queue.setAdjustment`.

`void _genesys.queue.resetAdjustment(_data.ixnid, target, statistic)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **target:** STRING which can be a variable or a constant - This parameter defines the target whose statistic adjustment is to be cleared. The string value of this parameter must be in the possible target format. See the Target Formats section for details.
- **statistic:** STRING which can be a variable or a constant - This parameter defines the name of the statistic for which the adjustment is to be removed.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.resetAdjustment(_data.ixnid,'SipGr_2.GA','StatAgentsAvailable');
```

_genesys.queue.routed

This function marks the interaction as routed. When routing functions, such as `<submit>` actions with the route attribute set to true, are successfully executed, an interaction is implicitly marked as routed. This function explicitly marks the interaction as routed. This function allows the functional module to dispose of the interaction when the strategy is completed and not route the interaction to the default destination.

`void _genesys.queue.routed(ixnid)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which routing has terminated.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.routed( _data.ixnid );
```

_genesys.queue.selectTargets

This function removes from a list of Agent Groups, Place Groups, or Queue Targets those targets that have already received a number of calls in excess of their quota for the interval when the function is called. These types of quotas are specified in the configuration layer in the *Statistical Days* belonging to *Statistical Tables* of type *Quota Table*, associated with each of the Agent Groups or Place Groups. The queue is associated with that *Quota Table* that is associated with an Agent Group for which the queue is an origination DN. If no agent group is found, the queue is associated with the *Quota Table* that is associated to an Agent Group named after the alias of the queue. If an Agent Group or Place Group target on the list has no *Quota Table* associated with it or no *Statistical Day* in the table matches the current date, the target is retained in the list returned by the function: it has not exceeded its quota since no quota was set for it.

Configuring Quota Tables Associated to Groups of Agents or Places

A *Quota Table* is configured in the configuration layer as a *Statistical Table* object of *Quota Table* type (see the configuration layer documentation for more details on the process of configuration). The*Quota Table* associated with an Agent Group or a Place Group must be specified inside the Advanced properties of the group. The same*Quota Table* can be associated with more than one Agent Group or Place Group. The *Quota Table* must contain *Statistical Days*. Use the information about *Statistical Days*, but not the information about Statistical Values. The relevant values for Statistical Values are as follows: *Statistical Value 1, Statistical Value 2*, and *Statistical Value 3* for each Interval of the day - during every interval. Value 1 is used when Quota type has a value of 0 (*QuotaMin*). Value 2 is used when Quota type has a value of 1 (*QuotaTarget*). Value 3 is used when Quota type has a value of 2 (*QuotaMax*). All other properties of Statistical Days are irrelevant for the purpose of setting up quotas. **Note:** The same *Statistical Day* can belong to more than one *Quota Table*.

`tlist _genesys.queue.selectTargets(ixnid, filter, targets)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **filter:** genesys.queue.quotaType ENUM OBJECT which can be a variable or a constant - This parameter indicates which of the three relevant entries in the statistical day will be treated as the current quota.
- **targets:** STRING which can be a variable or a constant - This parameter is a string of comma-separated high-level Agent Groups Or Place Groups or Queue Targets (in the possible target format, see the Target Formats section for details). If this parameter has queue targets, you need to make sure that every queue for which you use this parameter is listed as an origination DN for at most one Agent Group in the configuration layer. If more than one group or queue is associated with the same table, then the interactions routed to all of them are counted together. That is, the quota is interpreted as a limit on the total number of interactions routed to groups and queues associated with the same table. Therefore, you must set up individual Quota Tables for groups and queues that you want to consider separately, even if these tables consist of the same Statistical Days.

Returns: `tlist`: STRING - The result of the function is a string of targets in the possible Target Formats that matches the quota filter criteria. However, if the function encounters a list element in a different format, it does one of the following:

- If the element is a syntactically correct target in the complete high-level format but its type is not Group of Agents or Group of Places, the target is retained in the list returned by the function.
- If the element is not a syntactically correct target in the complete high-level format, including targets with an omitted type or location, it will be dropped from the list.

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.selectTargets( _data.ixnid, 1,
'SipGr_3@StatServer.GA,SipPlGr2@StatServer.GP' );
```

_data.myResult will be the following string:

```
'SipGr_3@StatServer.GA,SipPlGr2@StatServer.GP'
```

_genesys.queue.selectTargetsByThreshold

This function finds the best available target(s) from a list of targets by applying a statistic with a threshold comparison against the input target list. This function returns a subset of the `targets` List parameter (possibly empty).

```
tlist _genesys.queue.selectTargetsByThreshold(ixnid, targets, statistic, value, cond)
```

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **targets:** STRING which can be a variable or a constant - This parameter is a string of comma-separated high-level targets (in the possible target format, see the Target Formats section for details).
- **statistic:** STRING which can be a variable or a constant - This parameter is a statistic to be used in the comparison.
- **value:** NUMBER (INTEGER) which can be a variable or a constant - This parameter is the value that will be used in the comparison to see if a target meets the condition or not.
- **cond:** genesys.queue.statcond ENUM OBJECT which can be a variable or a constant - This parameter is the condition which is to be used in the comparison.

Returns: `tlist`: STRING - The result of the function is a string of targets in the possible Target Formats that matches the threshold filter criteria.

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.selectTargetsByThreshold( _data.ixnid,
'701_sip.A,702_sip.A,703_sip.A', 'StatCallsAnswered', 3, 0 );
```

_data.myResult will be the following string:

```
'702_sip@StatServer.A,703_sip@StatServer.A'
```

_genesys.queue.setInteractionAge

This function overrides the default age of an interaction from a routing perspective. This function can be useful if the age of the interaction will be used for placing interactions into waiting queues. The interaction age is the time accumulated since the interaction was known by Genesys (normally set at the very first resource or device the interaction enters).

By default, the interaction age is defined by the time of the last event (for example, `EventRouteRequest`). However, if an interaction is going to be routed more than once (for example, if an agent transfers an interaction on a routing point for re-routing or if the Voice Callback Universal Callback Server resubmits a callback interaction to URS), the time of the last event (for example, `EventRouteRequest`) is not always the best way to define interaction age.

```
void _genesys.queue.setInteractionAge(ixnid, keep)
```

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **keep:** BOOLEAN which can be a variable or a constant - This parameter indicates whether the default age algorithm should be. If this parameter is true, it fixes the current interaction age so that it does not depend on subsequent routing events. If the parameter is false, it unfixes the age of interaction so it again will be defined by the moment of the last event.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.setInteractionAge( _data.ixnid, true );
```

NOTE: **This method currently applies to voice interactions only.** For multimedia interactions the following can be employed to provide the equivalent information. Do not apply the following to voice interactions.

To set the interaction age at the time that the interaction is present, similar to keep set to true, for example within a script on the entry to the next state:

```
var timestamp = Math.floor((new Date().getTime())/1000 ) + ' 0';
var setIXNAge = { RouterData70 : '("t"="' + timestamp + '")' };
_genesys.ixn.setuData(setIXNAge);
```

For the equivalent of keep set to false:

```
   _genesys.ixn.deleteuData( "RouterData70" );
```

_genesys.queue.setAdjustment

This function enforces an adjustment of the values of a specified statistic for a particular target. The adjustment does not apply to the result of explicit Stat Server queries by the function's `_genesys.statistic.sData()` function. It is only used for thresholds, statistical interaction distribution, or when a statistic is supplied as an argument to the `<submit>` actions. Once set, an adjustment can be cleared by invoking the _genesys.queue.resetAdjustment function. This applies across all interactions and the corresponding `<submit>` actions for the session.

```
void _genesys.queue.setAdjustment(ixnid, target, statistic, sign, value)
```

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **target:** STRING which can be a variable or a constant - This parameter defines the target whose statistic adjustment is to be set. The string value of this parameter must be in the possible target format. See the Target Formats section for details.
- **statistic:** STRING which can be a variable or a constant - This parameter defines the name of the statistic which is to be adjusted.
- **sign:** STRING which can be a variable or a constant - This parameter defines the operation that will be applied to adjust the statistic. The following are the valid values:

    - +, the value is added to the result reported by Stat Server
    - -, the value is subtracted from the result reported by Stat Server
        - , the value is multiplied by the result reported by Stat Server
    - /, the result reported by Stat Server is divided by value.
- **value:** NUMBER (FLOAT) which can be a variable or a constant - This parameter defines the value that the statistic is to be adjusted by.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.setAdjustment( _data.ixnid, 'SipGr_2.GA', 'StatAgentsAvailable', '*', 2 );
```

_genesys.queue.translationOverride

This function overrides any translation specified in configured Switch Access Codes for routing to remote targets later in the session, and instructs the functional module to use the information specified in the parameters for the purpose of number translation.

`void _genesys.queue.translationOverride(ixnid, source, destination, location, rType, DNIS, reason, extension)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **source:** STRING which can be a variable or a constant - This parameter is the source device address to be used in the translation.
- **destination:** STRING which can be a variable or a constant - This parameter is the destination device address to be used in the translation.
- **location:** STRING which can be a variable or a constant - This parameter is the location of the source device to be used in the translation.
- **rType:** genesys.queue.rType ENUM Object which can be a variable or a constant - This parameter is the route type to be used in the translation.
- **DNIS:** STRING which can be a variable or a constant - This parameter is the DNIS to be used in the translation.
- **reason:** STRING which can be a variable or a constant - This parameter is the routing reason to be used in the translation.
- **extension:** STRING which can be a variable or a constant - This parameter is the extension information to be used in the translation.

Returns: `void`

The following is an example of input and output for this function:

```
_genesys.queue.translationOverride( _data.ixnid, 'source', 'dest', 'location', 1, '702',
'reason', 'ext' );
```

_genesys.queue.targetSelectionTuning

This function activates a configured cost-based routing solution for this session and the associated functional module objects (interactions and so on).

`void _genesys.queue.targetSelectionTuning(ixnid, useCostFactor)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **useCostFactor:** BOOLEAN which can be a variable or a constant - This parameter indicates whether cost factors should be used for making routing decisions for this session and the `<submit>` actions.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.targetSelectionTuning( _data.ixnid, true );
```

_genesys.queue.useAgentState

This function specifies the agent state the functional module will use instead of the default reported by Stat Server. This will apply to the entire session and all the `<submit>` actions until the next _genesys.queue.useAgentState invocation.

To use this option you must first set up URS as follows:

- In the *Annex* or *Options* tab of the platform application, create a section called AgentStates (case sensitive).
- Within that section, create an option for every user-defined agent state. The platform can accept up to 32 options in the *AgentStates* section.
- For each option, specify its value in the format:

```
Function[DNtype]<op1>Function[DNtype]<op1>...Function[DNtype]<op2>number...(Format1
expression)<op3>(Format1 expression)<op3>...
```

Where:

- `Function[DN type]` is one of the following predefined functions:
  - `ready[DN type]` - which returns the number of agent DNs of the specified type who are in the ready state at the current moment.
  - `busy [DN type]` - which returns the number of agent DNs of the specified type who are in the busy state at the current moment.

DN type for these predefined functions is the agent's DN. Types include *ACDPosition, Extension, E-mail, Eaport, Cellular, Chat, Cobrowse, Fax, Voicemail, Voip, Video*, and *Workflow*.

- op1 is an operator of either plus (+) or minus (-)
- op2 is an operator of either greater than (> - >), less than (< - <), or equal to (=)

- op3 is a logical operator of either or (|) or and (& - &)
- multiplication (*)
- division (/)
- number is zero or any positive number to evaluate the expression

For example, the agent is defined as ready if the agent has no calls on his or her extension or `position busy[extension]+busy[acdposition] = 0`. An agent in this state will be considered not ready if the agent has at least one call on the extension or position. The agent will be considered ready in all other situations. So, for example, if the agent has the e-mail DN busy, the agent is still considered ready.

Important Information

- When using the UseAgentState function, whole numbers are rounded (`1.25` is counted as `1`).
- Option values cannot contain spaces.
- If you want to use AgentStates for a backup functional module platform in addition to the primary platform, create an identical AgentStates section in the backup platform.
- When using this function, you must use lowercase DN types that do not include a "-" (hyphen). For example, use "email" instead of "E-mail".
- If function CheckAgentState is set to false, the functional module ignores any agent state, whether the default one (reported by Stat Server) or the user-defined one (as described above).
- The functional module uses integer arithmetic in its calculations, such as for agent state and skill expression evaluation. For this reason, you must always create expressions based on integer arithmetic, not floating point arithmetic.

`void _genesys.queue.useAgentState(ixnid, state)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **state:** STRING which can be a variable or a constant - This parameter is the agent state that should be used instead of the default reported from Stat Server.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.useAgentState( _data.ixnid, 'ready' );
```

_genesys.queue.useAgentStatistics

This function makes the functional module apply statistics for target selection at the level of individual Agents or Places even if the targets are groups of corresponding objects, such as Agent Groups or Place Groups.

This function may be used for any appropriate statistic. It can also be used for cost-based routing.

`void _genesys.queue.useAgentStatistics(ixnid, use)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **use:** BOOLEAN which can be a variable or a constant - This parameter indicates whether individual agent statistics should be applied. The default is false.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.useAgentStatistics( _data.ixnid, true );
```

_genesys.queue.useCapacity

This function instructs the functional module on the condition for using configured capacity tables for computing statistical values. If *OnStatError* is supplied to the parameter, the functional module will compute a statistical value from statistical tables whenever an attempt to obtain the corresponding value from Stat Server results in an error. The other two options are either always use statistical tables for such values or never use them.

`void _genesys.queue.useCapacity(ixnid, use)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **use:** genesys.queue.usecapcond ENUM OBJECT which can be a variable or a constant - This parameter indicates the method of computing statistical values.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.useCapacity( _data.ixnid, 1 );
```

_genesys.queue.useCustomType

This function instructs the functional module on the value of a target property that the target must to have to be considered a valid target for the current session and the associated interaction and the `<submit>` actions that are being used.

`void _genesys.queue.useCustomType(ixnid, tType, property, value)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **tType:** STRING which can be a variable or a constant - This parameter indicates the type of target to use. The following are the valid values:
  - "Agent" - An agent target

    - "DN" - A DN target
    - "Place" - A Place target
- **property:** STRING which can be a variable or a constant - This parameter indicates which property of the target will be used. The following is where these properties are defined and found:
    - "Agent" target type - The property is specified under the Switch object, inside the folder with the name of the platform application, in the Annex of the Agent Login. The functional module checks the agent type for the following targets: *Agents, Agent Groups, Places,* and *Place Groups.* For the last two targets, the agent type can be verified only if Stat Server reports the name of the agent associated with the Place in question.
    - "DN" target type - The property is specified under the Switch object, inside the folder with the name of the platform application, in the Annex of the DN. The functional module checks the DN type for the following targets: *Agents, Agent Groups, Places, Place Groups, Routing Points*, and *Queues*, as well as custom *DNs*.
    - "Place" target type - The property is specified in the Annex of Place inside the folder with the name of the platform application. The functional module checks the Place type for the following targets: *Agents, Agent Groups, Places*, and *Place Groups*.
- **value:** STRING which can be a variable or a constant - This parameter indicates the value of the target property which should be evaluated to determine if the target should be used for routing.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.useCustomType( _data.ixnid, 'DN', 'Extension', '702' );
```

_genesys.queue.useDNType

This function instructs the functional module on the type of DN to use when routing to a target. Choose the DN-based type of session and interaction being routed. This delivers various sessions and interactions to the correct DN on an agent's desktop.

Important information

- The functional module sets the default DN type from the *MediaType* attribute of the trigger event. This function overrides that default.

`void _genesys.queue.useDNType(ixnid, type)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **type:** genesys.resource.resourceType ENUM OBJECT which can be a variable or a constant - This parameter indicates the type of DN target to use.

Returns: `void`

The following is an example of input and output for this function:

```
_genesys.queue.useDNType( _data.ixnid, _genesys.resource.resourceType.CFGACDPosition );
```

_genesys.queue.useMediaType

This function instructs the functional module on the media types (for examples, voice, e-mail, or chat) that a target is associated with. In order to accept sessions and interactions of a particular media type, a target must be associated with that media type. If the interaction is not a voice interaction, the functional module sets the initial media type from the Media Type attribute of the trigger event. The UseMediaType function overrides this initial setting. The functional module can pick up for routing either the available media of an agent or a ready DN of an agent.

Important Information

- For backward compatibility, if some DN type (DNTYPE) has a corresponding media (MEDIA) associated with it (for example e-mail), then the `_genesys.queue.useMediaType` function is equivalent to `_genesys.queue.useDNType` function.
- This function indicates the functional module will select only Extensions or ACDPositions of the agent.

`void _genesys.queue.useMediaType(ixnid, type)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **type:** STRING which can be a variable or a constant - This parameter indicates the type of Media target to use. The set of valid values are from the media definitions in the configuration layer.

Returns: `void`

The following is an example of this function:

```
_genesys.queue.useMediaType( _data.ixnid, 'CFGEmail' );
```

_genesys.queue.expandGroup

This function creates a list of resources associated with a group. This list of (agent) resources can be used for the following purposes:

- To propagate a target-selecting statistic on the resource level.
- To allow the Queue functional module to handle situations in which a particular resource is a member of multiple groups and to solve related interaction-priority issues.

`resourcelist _genesys.queue.expandGroup(ixnid, group)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **group:** STRING which can be a variable or a constant - This parameter is a string in the Target Formats. It will be the name of an Agent Group (real or virtual) or a Place Group (`groupname@location.GA or groupname@location.GP`) **Note:** location is optional, type is mandatory.

Returns: `resourcelist`: STRING - The result of the function is a string which contains a comma-separated list of Agents or Place resources belonging to the specified Agent Group or Place Group. For example, `"agent1@StatServer1.A,...,agentN@StatServer1.A"`

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.expandGroup( _data.ixnid, 'SipGr_2.GA' );
```

_data.myResult will be the following string:

```
'701_sip@StatServer.A,702_sip@StatServer.A'
```

_genesys.queue.getSkillInGroup

This function returns the list of the agents belonging to the Agent Group based on the defined skill expression. You can also just specify a Stat Server and a skill expression without specifying an Agent Group. However, a skill expression and a Stat Server must be specified. The Stat Server is used to query the content of the provided Agent Group (real or virtual).

`resourcelist _genesys.queue.getSkillInGroup(ixnid, statserver, group, sexpr)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **statserver:** STRING which can be a variable or a constant - This parameter is optional. It is the name of the Stat Server containing information on the agents for this function. If not supplied, the Stat Server used will be the one defined by the *default_stat_server* configuration option of the platform.
- **group:** STRING which can be a variable or a constant - This parameter is optional. It is the agent group for the Stat Server that this function checks against. It can either be an agent group name or a list of comma-separated lists of agents, agent groups, places or place groups. Agents are included this group by either placing the agent name from the Persons folder into the Agent Groups folder or defining a virtual group using skill expression within the Annex tab of the Agent Group object.
- **sexpr:** STRING which can be a variable or a constant - This parameter defines the skill expression used to evaluate the agents. It can use skills, variables, numeric constants, and statistics to filter out agents based on their state. The statistic name in a skill expression can be any agent statistic used in the function `SData`. It must be written in the format: $(statistic). This ability to use statistics in a skill expression allows you to conduct queries based on a statistic. For example, if you want to query the number of agents with a Spanish skill of at least 5 who are currently logged in, the expression would be as follows: `Spanish >= 5 & $(StatAgentsLoggedIn)=1`.

Returns: `resourcelist`: String - The result of the function is a string which contains a comma-separated list of Agents or Place resources that meet the conditions of the skill expression. For example, `"agent1@StatServer1.A,...,agentN@StatServer1.A"`

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.getSkillInGroup( _data.ixnid, 'StatServer', 'SipGr_2',
'english>3' );
```

_data.myResult will be the following string:

```
'701_sip@StatServer.A,702_sip@StatServer.A'
```

_genesys.queue.expandActivity

This function creates a list of resources associated with a workforce management activity. It is intended for use with Genesys Workforce Management. It takes as a parameter a Workforce Management Activity name (defined in the configuration layer) and returns the list of resources assigned to the Activity from the current moment up to the next CutOffTime number of seconds. If a resource assigned to the Activity in the given time interval has a break of any kind (including assignment to another activity), that resource will not be included in the returned list.

`resourcelist _genesys.queue.expandActivity(ixnid, activity, cutoff)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **activity:** STRING which can be a variable or a constant - This parameter is a string. It will be the name of the workforce management activity to be used.
- **cutoff:** NUMBER which can be a variable or a constant - This parameter is the number representing the time in seconds that a resource has to be assigned to the activity starting from the current moment in order to be considered as qualified.

Returns: `resourcelist`: STRING - The result of the function is a string which contains a comma-separated list of agent resources belonging to the specified activity and meeting the cutoff time criteria. For example, `"agent1@StatServer1.A,...,agentN@StatServer1.A"` If the specified activity is not found, a null will be returned.

The following is an example of input and output for this function:

```
_data.myResult = _genesys.queue.expandActivity( _data.ixnid, 'taskA', 3 );
```

_data.myResult will be the following string:

```
'702_sip@StatServer.A'
```

_genesys.queue.resetTreatments (since 8.1.200.00)

This function clears the treatments associated with all outstanding `<submit>` actions associated with this session and can suspend the `<submit>` processing for the current session until the appropriate state has been reached to begin the next treatments.

When a list of busy treatments is used in a `<queue:submit>`, Universal Routing Server uses this list and plays the busy treatments one after another (by default, this list loops around). When the function `_genesys.queue.resetTreatments` is sent to Universal Routing Server, it will clear the list of busy treatments. However, if a treatment was being played while this function is received, the treatment currently playing will not be cleared. It is recommended to disable looping using `repeat` in the resource attribute for `<dialog:playsound>`.

`void _genesys.queue.resttreatments(ixnid)`

Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.

Returns: `void`

The following is an example of this function being used:

```
<state id="findAgent">
        <onentry>
                <queue:submit priority="5" timeout="60">
                        <queue:targets>
                                <queue:target type="agent" name="'agent1'" />
                        </queue:targets>
                        <dialog:runtreatments>
                                <dialog:playsound type="'music'" resource="'music/
on_hold;repeat=1'" duration="15" />
                                <dialog:playsound type="'music'" resource="'music/
in_queue;repeat=1'" duration="15" />
                        </dialog:runtreatments>
                </queue:submit>
        </onentry>
        <transition event="stopMusic" target="resetTreatments" />
        <transition event="queue.submit.done" target="state2" />
        <transition event="error.queue.submit" target="state3" />
</state>
<state id="resetTreatments">
        <onentry>
                <script>
                        _data.myResult = _genesys.queue.resetTreatments(_data.ixnid);
                </script>
        </onentry>
</state>
```

If the `stopMusic` event is received within the first 15 seconds while the `music/on_hold` is playing, it will play until the end of the file (ignoring the duration attribute), and `music/in_queue` will never be played because `resetTreatments` has cleared it from the list.

If the `stopMusic` event is received 50 seconds after entering the state `findAgent`, `music/in_queue` will be playing when the event is received, and the file will play until the end (igoring the duration attribute). No other music file will be played after this.

_genesys.queue.setDNIS

This function replaces the DNIS of the interaction by the appropriate value. It will override any previous`_genesys.queue.setDNIS()` function invocations.

Important Note: In order to get this DNIS override into the actual interaction on the controlling server (T-Server), the session- and interaction-originating event (for example, `EventRouteRequest`) must have had the routing type of *RouteTypeOverwriteDNIS* to make the change in the server. Otherwise, the override is only set within the session and its associated interaction object.

`void _genesys.queue.setDNIS(ixn, type, value)`

Parameters:

- **ixn:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have its DNIS overridden.
- **type:** genesys.queue.overwriteType ENUM OBJECT which can be a variable or a constant - This parameter defines the source of the value to set the DNIS to.

- **value:** STRING which can be a variable or a constant - This parameter defines the value that the DNIS is to be set to. This parameter is only valid when the type parameter is set to "UseValue".

Returns: `void`

The following is an example of input and output for this function:

```
_genesys.queue.setDNIS( _data.ixnid, 2, '3318' );
```

_genesys.queue.useMediaChannel (since ORS 8.1.200.48 and URS 8.1.200.25)

This function provides information about the type of controller that owns the interaction (either T-Server/SIP-Server or Interaction Server). In most cases, Queue FM itself defines the right controller type and there is no need to use this function explicitly. However, some types of media interactions (like chats) can be managed by either type of controller and Queue FM needs to be made aware of the exact controller type.

If Queue FM is called in "interactionless" context (when ixnid is set to null), Queue FM assumes T-Server/SIP-Server as the default controller type. If this is not the case, function `useMediaChannel` can be used to change this default.

`void _genesys.queue.useMediaChannel(ixnid, enable)`

Parameters (all parameters are mandatory):

- **ixnid:** STRING which can be a variable or a constant - This parameter defines the ID of the interaction which should have this action applied.
- **enable:** BOOLEAN which can be a variable or a constant - This parameter defines whether the interaction is controlled by T-Server/SIP-Server (if it is set to false) or Interaction Server (if it is set to true).

Returns: `void`

The following is an example of this function. When an SCXML session is started through Web API and Queue FM is used to find an agent ready to accept chats controlled by Interaction Server, the following should be invoked:

```
_genesys.queue.useMediaChannel(null, true);
```

Parameter Elements

The `<submit>` action element has parameter elements that can be used as input for the target and outbound attributes.

<targets>

This is the top-level element which defines the set of targets from which a given target is selected for the submit request.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
type	false	NMTOKEN	none	agent, place, agentgroup, placegroup, queue, dn, skill, campaigngroup, routepoint, label	This specifies the default resource type that should be used if a type attribute is not specified in the associated `<target>` elements.
statserver	false	value expression	none	any value expression that returns a valid string	A value expression which returns the default statserver that should be used if a statserver attribute is not specified in the associated `<target>` elements. See SCXML Legal Data Values and Value Expressions for details.

Children

- `<target>` Occurs 0 to N
- `<targetid>` Occurs 0 to N
- `<activity>` Occurs 0 to 1 - This element is also mutually exclusive with the other child elements.
- `<workbin>` Occurs 0 to 1 - This element is used as the target's workbin of objects specified in the `<targets>` element.

<targetid>

This defines a specific means of representing a set of targets.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
expr	true	value expression	none	Any value expression that returns a string that follows the format defined in the description	This is the ID of the target that is to be used. It is a string with a set of comma-separated sub-strings with the following

Name	Required	Type	Default Value	Valid Values	Description
					format: `{threshold}[weight]name@server.t` or `{threshold}[weight]?name:skille` The following is an example: `"'{StatCallsInQueue <10}[25]8001.Q, {StatCallsInQueue < 10}[25]8002.Q, {StatCallsInQueue < 20}[50]8003.Q"'` The following are the details of the different elements of the sub-strings. • weight - same as defined in `<target>` element • name - same as defined in `<target>` element • server - same as defined in statserver attribute in the `<target>` element • type - same as defined in `<target>` element, but with the following abbreviations: • A (for agent), • AP (for agent place) • GA (for

Name	Required	Type	Default Value	Valid Values	Description
					group of agents) • GP (for group of places) • Q (for Queues) • DN (for dn) • GA (for skill) • GC (for campaign group) • RP (for routing points) • DL (for destination label) • skilexpr - same as defined in `<target>` element • threshold - same as defined in `<target>` element See SCXML Legal Data Values and Value Expressions for details.

Children

None

<target>

This defines the resource criteria for selecting a target.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
name	false	value expression	none	Any value expression that returns a valid string which represents a resource of the defined type.	A value expression which returns the name of the target that is to be used. See SCXML Legal Data Values and Value Expressions for details.
skillexpr	false	value expression	none	Any value expression that returns a valid string which represents a valid skill expression	This is the skill expression associated with this `<target>` element. For details on the format, see the Skill Expressions section. See SCXML Legal Data Values and Value Expressions for details.
type	false	NMTOKEN	none	agent, place, agentgroup, placegroup, queue, dn, skill, campaigngroup, routepoint, label	This specifies the resource type associated with this `<target>` element. The skill and agentgroup types are synonymous.
statserver	false	value expression	none	Any value expression that returns a valid string which represents a valid stat server.	A value expression which returns the statserver that should be used for this target definition. See SCXML Legal Data Values and Value Expressions for details.
threshold	false	value expression	none	Any value expression that returns a valid string which represents a valid threshold expression	A value expression which returns a criteria definition that is used to further filter the potential possible targets associated with this `<target>` attribute. threshold is an analog of the strategy function

Name	Required	Type	Default Value	Valid Values	Description
					SetTargetThreshold and defines additional conditions the target must meet to be considered as valid target for routing. The following queue-specific methods can be used in a value expression. These methods are not executed inline as part of interpreting this attribute but are processed by the underlying queue functional module: • **`sdata(target, statistic)`**- Use this function to affect routing conditions based on statistics. • **`Acfgdata(Application name, folder, property, default value)`** - Use this function to affect routing conditions based on external data stored in properties of configuration layer application objects (ApplicationConFigDATA). • **`Lcfgdata(list name, folder, property,`**

Name	Required	Type	Default Value	Valid Values	Description
					`default value)` - Use this function to affect routing conditions based on external data stored in IRD list objects. • **`callage[]`** - Use this function to return the age of an interaction in seconds. See SCXML Legal Data Values and Value Expressions for details. **Note:** Orchestration evaluates this expression to create a string that is sent to Universal Routing Server. That is, variables and functions available in ORS may be used to construct this string; however URS has no notion of these values. The resultant string must be a completely self contained expression for URS. In other words, if you have a variable *x* in your strategy, and you want to use callage[] < *x*, it should be written as: 'callage[] < ' + *x*.
weight	false	value expression	none	Any value expression that returns a valid string	A value expression which returns a value that defines the weight of this `<target>` element against other

Name	Required	Type	Default Value	Valid Values	Description
					`<target>` elements. See SCXML Legal Data Values and Value Expressions for details.

- The name, skillexpr, weight, statserver and type attributes are interpreted as constants. They are components of the target specification format `[weight]name@statserver.type`. If skillexpr is present, then the format is `[weight]?name:skillexpression@statserver.type`.

Children

None

<activity>

This defines the workforce management activity criteria for selecting a target.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
name	true	value expression	none	Any value expression that returns a valid string which represents the name of a WFM activity	A value expression which returns the name of a WFM activity that is to be used for target selection. See SCXML Legal Data Values and Value Expressions for details.
cutofftime	false	value expression	none	Any value expression that returns an integer	A value expression which returns the cutoff time in seconds. This defines the window time in which a potential target resource must be assigned to the given activity. See SCXML Legal Data Values and Value Expressions for details.

Children

None

<workbin>

This defines the workbin criteria for selecting a target.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
type	true	NMTOKEN	none	agent, place, agentgroup, placegroup	This specifies the resource type associated `<workbin>` element.
name	true	value expression	none	Any value expression that returns a valid string which represents a workbin.	A value expression which returns the name of the workbin that is to be used. See SCXML Legal Data Values and Value Expressions for details.
loggedinonly	false	boolean expression	false	Any value expression that returns true or false	A boolean expression which returns whether logged out agents can pull interactions. (Dis)Allow using of logged out agents.

Children

None

Action Elements

<submit>

This action queues the request for a target based on the criteria specified in the request.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated integer identifier (based on URS-3372 changed from unique string) to be associated with the action being sent. This value will only be valid when the queue.submit.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.
queue	false	value expression	none	Any value expression which returns a valid string	A value expression which returns the name of the (virtual) queue that this request should be put in. See SCXML Legal Data Values and Value Expressions for details.
priority	false	value expression	0	Any value	A value expression

Name	Required	Type	Default Value	Valid Values	Description
				expression which returns a value integer	which returns the priority that the interaction will be given in the queue. See SCXML Legal Data Values and Value Expressions for details.
ordertype	false	value expression	any	max, min, any, percentage	A value expression which returns how type of ordering that should be used on the targets. It is used together with **orderstat**. See SCXML Legal Data Values and Value Expressions for details.
orderstat	false	value expression	none	Any value expression which returns a valid string	A value expression which returns the name of the statistic that will be used as a target selection criterion for ordering the targets. No orderstat specified means any target and order. See SCXML Legal Data Values and Value Expressions for details.
interactionid	false	value expression	none	Any value expression which returns a valid string	A value expression which returns the _genesys.ixn.interactions[x].g_uid associated with this request. There is a special value that can be returned: • ECMAScript **Null** means the functional module will not use an interaction for the request. **Note:** if the

Name	Required	Type	Default Value	Valid Values	Description
					`<outbound>` element is present then this attribute is ignored because a new interaction will be created. See SCXML Legal Data Values and Value Expressions for details.
route	false	boolean expression	true	Any expression which returns a boolean (true, false)	A boolean expression which returns whether or not this action should also redirect the interaction to the selected destination. There are two values that can be returned: • **"false"** means the functional module will not attempt to route the associated interaction. • **"true"** means the functional module will use the associated interaction to route the interaction. **Note:** a value of "true" is only supported for voice-related interactions. See SCXML Conditional Expressions for details.
clearontimeout	false	boolean expression	true	Any expression	A boolean

Name	Required	Type	Default Value	Valid Values	Description
				which returns a boolean (true, false)	expression which indicates whether or not the request and all associated `<submit>` requests for this interaction and queue should be removed from the queue after the timeout of this request. See SCXML Conditional Expressions for details.
timeout	false	value expression	0	A value expression which returns an integer	A value expression which returns an integer that represents the number of seconds to wait. See SCXML Legal Data Values and Value Expressions for details. The integer returned must be interpreted as a time interval. This interval begins when `<submit>` is executed. A failed and timed-out submit must return the error.queue.submit event. If the `<outbound>` element is present, this attribute is only used while waiting for an agent to become available and not for the request in its entirety.
threshold	false	value expression	none	Any value expression that returns a valid string which represents a valid threshold expression.	A value expression which returns a criteria definition that is used to further filter the potential possible targets associated with the queue attribute. **threshold** is an analog of the strategy function SetVQTargetThreshold and defines

Name	Required	Type	Default Value	Valid Values	Description
					additional conditions the target must meet to be considered as a valid target for routing with this queue. The following queue-specific methods can be used in a value expression. These methods are not executed inline as part of interpreting this attribute but are processed by the underlying queue functional module: • **sdata**(target, statistic) - Use this function to affect routing conditions based on statistics. • **acfgdata**(Application name, folder, property, default value) - Use this function to affect routing conditions based on external data stored in properties of configuration layer application objects (ApplicationConFigDATA). • **lcfgdata**(list name, folder, property,

Name	Required	Type	Default Value	Valid Values	Description
					default value) - Use this function to affect routing conditions based on external data stored in IRD list objects. • **callage** function - Use this function to return the age of an interaction in seconds. See SCXML Legal Data Values and Value Expressions for details.
src	false	value expression	none	A value expression which returns one of the following valid URI schemes: • gdata	This allows a developer to supply a URI which identifies the location of a `<submit>` definition that is to be used in the application. This attribute is mutually exclusive with the following attributes: • queue • ordertype • orderstat • timeout This attribute is also mutually exclusive with the children of this element. This source definition will replace the entire `<submit>` element in the application at load

Name	Required	Type	Default Value	Valid Values	Description
					time. See SCXML Legal Data Values and Value Expressions for details.

Other Considerations

- All events generated from the imbedded treatment actions will be suppressed as part of the action.
- The `<submit>` action does not work for targets such as multimedia queues. The `<redirect>` action is a way to place interactions into those queues.
- The **src** attribute supports only the gdata scheme, which maps to routing rules of type:
  - Statistic
  - Load Balance
  - Percentage
  - Workforce
- The route=true attribute value is supported only for voice-related interactions. For all others, route= false must be used with a subsequent `<redirect>` action. The error.queue.submit will be generated if this attribute is implicitly or explicitly set to true for all other interaction media types.
- The following rules are applied to the interactionid parameter in this action as well as in any other from the Queue functional module:
  - Genesys recommends that you always specify it explicitly.
  - If the `<scxml>` document has _type=routingstrategy and there are no interactions associated with the session (for example, a session is started through web interfaces), the special value null can be used. This will result in the Queue functional module using a platform-specific interactionid for communication purposes.

The following are examples of different types of target selection that can be done with `<submit>` and SCXML:

- **Load Balancing**

As you can see if you want to perform load balance functionality similar to the load balancing function block, you would use one of the load balancing statistics (this is using the basic one). In addition, you can really only load balance on DN-related resources (queue and dn types).

```
<state id="load_balancing">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:submit requestid="_data.reqid" orderstat="'StatLoadBalance'" ordertype="'min'"
timeout="100" >
    <queue:targets type="queue">
      <queue:target name="'queue1'"/>
      <queue:target name="'queue2'"/>
    </queue:targets>
  </queue:submit>
</onentry>
<transition event="queue.submit.done" target="statex">
  <assign location="target" expr="_event.data.targetselected"/>
</transition>
<transition event="error.queue.submit" target="statey"/>
</state>
```

- **Percentage**

As you can see, if you want to perform percentage routing functionality similar to the percentage function block, you set the ordertype attribute to "percentage" and set the appropriate target weights.

```
<state id="percentage">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:submit requestid="_data.reqid" ordertype="'percentage'" timeout="100" >
    <queue:targets type="agentgroup">
      <queue:target name="'agtgrp1'" weight="'20'"/>
      <queue:target name="'agtgrp2'" weight="'80'"/>
    </queue:targets>
  </queue:submit>
</onentry>
<transition event="queue.submit.done" target="statex"/>
<transition event="error.queue.submit" target="statey"/>
</state>
```

- **Selection (Generic Target Selection)**

This type of target selection is generic and can be used to combine several different forms of target selections. The biggest difference is the ability to define thresholds for a given target definition. This example shows how to do that.

```
<state id="Selection">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:submit queue="'VQ1'" requestid="_data.reqid" ordertype="'max'"
orderstat="'StatTimeInReadyState'" timeout="100">
    <queue:targets type="agentgroup" statserver="'www.genesyslab.stserver1.com'">
      <queue:target name="'agtgrp1'" threshold= "'sdata(agtgrp1.GA, StatAgentsAvailable) >
10'"/>
      <queue:target name="'agtgrp2'"/>
    </queue:targets>
  </queue:submit>
</onentry>
<transition event="queue.submit.done" target="statex"/>
<transition event="error.queue.submit" target="statey"/>
</state>
```

- **Statistics**

As you can see here, if you want to perform statistical routing functionality similar to the statistics function block, you just have to specify the ordertype and orderstat on the action and either specify a single statserver for all targets or a specific one for each to do target selection.

```
<state id="Statistics">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:submit queue="'VQ1'" requestid="_data.reqid" ordertype="'min'"
orderstat="'StatExpectedWaitingTime'" timeout="100">
    <queue:targets type="agentgroup" statserver="'www.genesyslab.stserver1.com'">
      <queue:target name="'agtgrp1'"/>
      <queue:target name="'agtgrp2'"/>
    </queue:targets>
  </queue:submit>
</onentry>
<transition event="queue.submit.done" target="statex"/>
<transition event="error.queue.submit" target="statey"/>
</state>
```

- **Skills-Based**

As you can see here, if you want to perform skills-based routing functionality similar to that which is available on several function blocks, you just have to specify the skillexpr attribute on a target element. This can be mixed with other targets as well.

```
<state id="Skills-based">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:submit queue="'VQ1'" requestid="_data.reqid" ordertype="'min'"
orderstat="'StatExpectedWaitingTime'" timeout="100">
    <queue:targets type="agentgroup" statserver="'www.genesyslab.stserver1.com'">
      <queue:target skillexpr="'service1 > 5 & english > 3'"/>
      <queue:target name="'agtgrp2'"/>
    </queue:targets>
  </queue:submit>
</onentry>
<transition event="queue.submit.done" target="statex"/>
<transition event="error.queue.submit" target="statey"/>
</state>
```

- **Using Busy Treatments**

As you can see here, if you want to perform treatments in conjunction with routing functionality you need to specify the appropriate `<runtreatments>` element defining which treatments are to run while waiting for a target to be found.

```
<state id="Skills-based_with_treatments">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:submit queue="'VQ1'" requestid="_data.reqid" ordertype="'min'"
orderstat="'StatExpectedWaitingTime'" timeout="100">
    <queue:targets type="agentgroup" statserver="'www.genesyslab.stserver1.com'">
      <queue:target skillexpr="'service1 > 5 & english > 3'"/>
      <queue:target name="'agtgrp2'"/>
    </queue:targets>
    <dialog:runtreatments>
```

```
      <dialog:playsound type="'music'" resource="'EMusicDN'" duration="100"/>
      <dialog:play language="'English (US)'" >
        <dialog:prompts type="'tts'">
          <dialog:prompt interrupt="true" text="'The estimated wait time is' +
_genesys.statistic.sData('agtgrp2@.GA', 'StatExpectedWaitingTime')"/>
        </dialog:prompts>
      </dialog:play>
    </dialog:runtreatments>
  </queue:submit>
</onentry>
<transition event="queue.submit.done" target="statex"/>
<transition event="error.queue.submit" target="statey"/>
</state>
```

- **Activity-Based**

As you can see here, if you want to perform activity(WFM)-based routing functionality similar to that which is available on several function blocks, you just have to specify the `<activity>` element in the `<targets>` element.

```
<state id="Activity-based">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:submit queue="'VQ1'" requestid="_data.reqid" ordertype="'min'"
orderstat="'StatExpectedWaitingTime'" timeout="100">
    <queue:targets statserver="'www.genesyslab.stserver1.com'">
      <queue:activity name="'service1'"/>
    </queue:targets>
  </queue:submit>
</onentry>
<transition event="queue.submit.done" target="statex"/>
<transition event="error.queue.submit" target="statey"/>
</state>
```

- **Using Routing Rules From Configuration Server**

As you can see here, if you want to perform routing rule routing functionality similar to the existing routing function block, you just have to specify the src attribute using the gdata URI scheme.

```
<state id="RoutingRules_from_Configuration_Server">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:submit src="gdata:routingrule1"/>
</onentry>
<transition event="queue.submit.done" target="statex"/>
<transition event="error.queue.submit" target="statey"/>
</state>
```

Children

- `<targets>` - Occurs 1 time. This instance defines a named set of targets.
- `<runtreatments>` - Occurs 0 or 1 times. This instance defines a set of treatments to apply while this request is queued. The treatment-related events will not be generated as part of this request. The treatment can have an additional parameter, duration (if it doesn't have it yet). When the treatment

expires, the next treatment in the definition will be applied. If treatments that collect digits are used, the application will only get the digits collected by the last treatment that collected the digits when this action has ended (successfully or unsuccessfully). These digits will be available to the application via the _event.data.digits property.

Events

The following events can be generated as part of this action:

- `queue.submit.done`
- `queue.submit.requestid`
- `error.queue.submit` - This event will be sent as a result of the following conditions:
  - A timeout of the request
  - Problems with the request itself.
  - The customer abandons the interaction
  - The platform failed to contact the customer based on the criteria specified.
  - The platform failed to contact the agent that was selected.
- `queue.cancelled` - This event will be sent either when a target was selected from another outstanding `<submit>` action, in which case it indicates that this `<submit>` action request was cancelled, or when a `<submit>` action request has been cancelled using the `<cancel>` action.
- `interaction.deleted` - This event will be sent when the customer abandons the interaction associated with this `<submit>` request.

Note: For every `<queue.submit>` action, one and only one event can be created with the reference id generated with this submit.action. Specifically:

- `queue.submit.done` - One of the targets from this queue was selected.
- `error.queue.submit` - The submit action fails to apply.
- `queue.cancel.done` - The interaction was explicitly requested to be removed specifically from the queue with this request id.
- `queue.cancelled` - The interaction was removed from the queue as a side effect of another successfully completed action.

<cancel>

This action removes the requests from the queue and from consideration as targets. This is equivalent to doing another `<submit>` with the clearqueue attribute set to true or to issuing the IRD function ClearTargets.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location	This is the location

Name	Required	Type	Default Value	Valid Values	Description
				expression	of the ID of the outstanding request which is to be canceled. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details.
queue	false	value expression	none	Any value expression which returns a valid string	A value expression which returns the name of the (virtual) queue that this request is for. If there was more than one `<submit>` request for the same interaction and this queue, then all these requests will be removed from this queue and from consideration as a target. See SCXML Legal Data Values and Value Expressions for details.
interactionid	false	value expression	none	Any value expression which returns a valid string	A value expression which returns the _genesys.ixn.interactions[x].g_uid associated with this request. This is only used in conjunction with the queue attribute. There is a special value that can be returned: • ECMAScript **Null** means the functional module will not use an interaction for the request. See SCXML Legal Data Values and Value Expressions for details.

The requestid and queue attributes are mutually exclusive. If the requestid is present, then just that request is cleared from the queue and from target selection.

If the queue is present, then all outstanding requests are cleared from the defined queue. As part of this processing, the appropriate queue.cancelled events will be fired for all requests that are cleared.

The interactionid attribute is only used in conjunction with the queue attribute and is only needed when your application is handling multiple interactions.

The following are some examples:

```
<state id="cancel">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:cancel requestid="_data.reqid"/>
</onentry>
<transition event="queue.cancel.done" target="statex"/>
<transition event="error.queue.cancel" target="statey"/>
</state>
<state id="cancel">
<onentry>
  <queue:cancel queue="'vq1'" />
</onentry>
<transition event="queue.cancel.done" target="statex"/>
<transition event="error.queue.cancel" target="statey"/>
</state>
<state id="cancel">
<datamodel>
  <data id="ixnid"/>
</datamodel>
<onentry>
  <queue:cancel queue="'vq1'" interactionid="_data.ixnid" />
</onentry>
<transition event="queue.cancel.done" target="statex"/>
<transition event="error.queue.cancel" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `queue.cancel.done`
- `error.queue.cancel`
- `queue.cancelled` - This event is sent for all requests that are cleared.

Note: The `queue.cancelled` events will be sent before the `queue.clear.done`.

<update>

This action updates the criteria associated with an outstanding submit request.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	True	location expression	none	Any valid location expression	This is the location of the ID of the outstanding request which is to be updated. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details.
interactionid	False	value expression	none	Any value expression which returns a valid string	A value expression which returns the _genesys.ixn.interactions[x].g_uid associated with this request. If the interactionid attribute value is not associated with an outstanding and corresponding `<submit>` action, then an error event (error.queue.cancelled) will be generated. There is a special value that can be returned: • ECMAScript **Null** means the functional module will not use an interaction for the request. See SCXML Legal Data Values and Value Expressions for details.
priority	False	value expression	none	Any value expression that returns a valid integer	A value expression which returns the priority that the interaction will be assigned in the

Name	Required	Type	Default Value	Valid Values	Description
					queue. See SCXML Legal Data Values and Value Expressions for details.
timeout	False	value expression	none	A value expression which returns an integer	A value expression which returns an integer that represents the number of seconds to wait. See SCXML Legal Data Values and Value Expressions for details. The integer returned must be interpreted as a time interval. This interval begins when `<update>` is executed. A failed and timed-out submit must return the error.queue.update event. If the original submit request an outbound one, this attribute is only used while waiting for an agent to become available and not for the request in its entirety.

The following are examples:

This example updates the timeout value for this request:

```
<state id="update">
<datamodel>
  <data id="reqid"/>
  <data id="ixnid"/>
</datamodel>
<onentry>
  <queue:update requestid="_data.reqid" interactionid="_data.ixnid" timeout="300"/>
</onentry>
<transition event="queue.update.done" target="statex"/>
<transition event="error.queue.update" target="statey"/>
</state>
```

This example updates the priority for this request:

```
<state id="update">
<datamodel>
```

```
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:update requestid="_data.reqid" priority="6" />
</onentry>
<transition event="queue.update.done" target="statex"/>
<transition event="error.queue.update" target="statey"/>
</state>
```

This example updates the set of targets to select from for this request:

```
<state id="update">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:update requestid="_data.reqid"/>
    <queue:targets>
      <queue:target skillexpr="'service1 > 5 & english > 3'"/>
    </queue:targets>
  </queue/update>
</onentry>
<transition event="queue.update.done" target="statex"/>
<transition event="error.queue.update" target="statey"/>
</state>
```

This example updates the treatments of this request:

```
<state id="update">
<datamodel>
  <data id="reqid"/>
</datamodel>
<onentry>
  <queue:update requestid="_data.reqid">
    <dialog:runtreatments>
      <dialog:playsound type="'music'" resource="'EMusicDN'" duration="'100'"/>
    </dialog:runtreatments>
  </queue:update>
</onentry>
<transition event="queue.update.done" target="statex"/>
<transition event="error.queue.update" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `queue.update.done`
- `error.queue.update`

<query>

This action queries the status of the request. It returns the following information in the `queue.query.done` event for a specific interaction:

- priority - This is the current priority of the request.
- positioninqueue - This is the current position of the request in the queue.
- Invqwaittime - This is the expected wait time for the queue in relationship to the request.

This action can be used in two cases:

- While `<submit>` requests are outstanding, to determine what to do next. For example, while the application waits for a target, if the customer presses the "I cannot wait anymore" number, this event is processed by the application (`<transition>`). The application queries the status of the request to determine what type of treatment to present to the customer. If the customer has been waiting 10 minutes and the queue depth has not changed then the application may offer the customer the option of having a callback set up for him or her (using a dialog or treatment).
- After a `<submit>` request has timed out, if the interaction was not cleared from target selection. The application may need to collect information on where this interaction stands with respect to it still being in the queue. This information would then be used to determine what to do next (for example, provide the customer with other options, do another `<submit>` but with different target selection criteria (other targets, a new queue, and so on).

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	true	location expression	none	Any valid location expression	This is the location of the ID of the outstanding request which is to be queried. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details.
interactionid	false	value expression	none	Any value expression which returns a valid string	A value expression which returns the _genesys.ixn.interactions[x].g_uid associated with this request. If the interactionid attribute value is not associated with the outstanding corresponding `<submit>` action, then an error event (error.queue.query) will be generated. There is a special value that can be returned: • ECMAScript

Name	Required	Type	Default Value	Valid Values	Description
					Null means the functional module will not use an interaction for the request. See SCXML Legal Data Values and Value Expressions for details.

The interactionid attribute is only used when your application is handling multiple interactions.

The following is an example:

```
<state id="Query">
<onentry>
  <queue:query requestid="_data.reqid">
</onentry>
<transition event="queue.query.done" target="statex">
  <if cond="_event.data.positioninqueue > 200">
    <queue:update requestid="_data.reqid" priority="_event.data.priority + 5"/>
  </if>
</transition>
<transition event="error.queue.query" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `queue.query.done`
- `error.queue.query`

<default>

This action will redirect the interaction to its associated default destination or may optionally returns the configured default target address with out redirecting the interaction. This may be used when the orchestration logic cannot find a suitable destination to redirect the interaction to.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the queue.default.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	false	value expression	none	A valid value expression	A value expression which returns the _genesys.ixn.interactions[x].g_uid associated with this request. There is a special value that can be returned: • ECMAScript **Null** means the functional module will not use an

Name	Required	Type	Default Value	Valid Values	Description
					interaction for the request. See SCXML Legal Data Values and Value Expressions for details.
route	false	boolean expression	true	Any expression that returns a boolean (true, false)	A boolean expression which returns true or false. The meaning of which implies the following: • **"false"** means the functional module will not attempt to route the associated interaction and only the default destination associated will be returned in the `queue.default.done` event. • **"true"** means the functional module will use the associated interaction to route the interaction. **Note:** a value of "true" is only supported for voice-related interactions. See SCXML Conditional Expressions for

Name	Required	Type	Default Value	Valid Values	Description
					details.

The following is an example:

```
<state id="Default">
  <onentry>
    <queue:default/>
  </onentry>
  <transition event="queue.default.done" target="statex"/>
  <transition event="error.queue.default" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `queue.default.done`
- `queue.default.requestid`
- `error.queue.default`

Events

The following are the Queue action result events:

Event	Attributes	Description
queue.submit.done		This event indicates the success of the request and that a target has been selected.
	requestid	This is the ID associated with the request.
	targetselected	This is the DN and the switch name of the target to which the interaction was routed or should be routed to definitively; the target format is (`Name@SwitchName.Type`).
	vqselected	This is the virtual queue that was selected.

Event	Attributes	Description
	targetcomponentselected	This is the agent-level target to which the interaction was routed or should be routed to definitively. If the target specified in `<submit>` and selected for routing is of type Agent, Place, Queue, or Routing Point, this contains the target itself. If the desired target type is Agent Group, Place Group, or Queue Group, the function returns the agent, place, or queue from the corresponding group the interaction was sent to. The target format is (`Name@StatServerName.Type`).
	targetobjectselected	This is the high-level target (one that you specify in a `<submit>`) to which the interaction was routed or should be routed to definitively. If a skill expression is used, the function returns: `?:SkillExpression@statserver.GA` or even `?GroupName:SkillExpression@statserver.GA` The target format is (`Name@StatServerName.Type`).
	resource	This is an ECMAScript resource object which represents the target selected and can be used on any interaction-related action. See the Resource object for details.
	access	This attribute is optional and provided only if <submit> parameter route set to false and siwtch access code is defined between source and destination swithces for target type that match type of selected target. When present it is an ECMAScript object which represents switch access code and has following sert of properties: prefix, rtype, destination, location and dnis. Their values match to following switch access code fileds: Code, Route Type, Destination Source, Location Source and DNIS Source.
queue.submit.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.queue.submit		This indicates that an abnormal condition occurred while trying to perform the

Event	Attributes	Description
		request. This event will be sent as a result of a timeout of the request, as well as problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • -001 Timeout • 0001 Unknown error • 0008 Routing done • 0013 Remote error • 0018 Unknown object • 0019 Translation failed
	description	if error parameter has value '0013 Remote error' then this field might contain additional error information as provided by call/interaction controller (for example by TServer). Format of this string field is 'ErrorNumber ErrorMessage'
queue.cancel.done		This event indicates the success of the clear request and that the request has been removed from the queue.
	requestid	This is the ID of the `<cancel>` request.
error.queue.cancel		This indicates that an error occurred while trying to perform the `<cancel>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred.
	description	This is a more detailed description of the error.
queue.update.done		This event indicates that the request has been updated successfully.

Event	Attributes	Description
	requestid	This is the ID of the `<update>` request.
error.queue.update		This indicates that an error occurred while trying to perform the `<update>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • Invalidrequestid
	description	This is a more detailed description of the error. The following are the possible values: • Invalidrequestid - The request id xxxx does not match any outstanding `<submit>` requests. xxxx is the value of the requestid attribute.
queue.query.done		This event indicates the success of the query request.
	requestid	This is the ID of the `<query>` request.
	priority	This is the current priority of the request.
	positioninqueue	This is the current position of the request in the queue.
	invqwaittime	This is the expected wait for the queue in relationship to the request.
	totalsize	This is the total number of agents that can be targetted for the request. (Only since URS 8.1.3)
	loginsize	This is the number of agents logged in that can be targetted for the request. (Only since URS 8.1.3)
error.queue.query		This indicates that an error occurred while trying to perform the `<query>` request.

Event	Attributes	Description
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred.
	description	This is a more detailed description of the error
queue.default.done		This event indicates the success of the request and that a default target has been selected.
	requestid	This is the ID associated with the request.
	defaultselected	This is the default configured target address.
queue.default.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.queue.default		This indicates that an error occurred while trying to perform the `<default>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred: • timeout • invalidattribute • abandoned • unknown • invalidstate.state (null, ringing, hold, transferring, treating, routed) • badtranslation • remote
	description	This is a more detailed description of the

Event	Attributes	Description
		error. The following are the possible values: • timeout - the interaction was not redirected to the default target in a timely manner. • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • abandoned - The customer has abandoned the interaction. • unknown - The cause of the failure is unknown. • invalidstate.state - The interaction is in an invalid state and cannot be redirected to the default target. • badtranslation - The destination address xxx could not be translated. xxx is the address of the default target. • remote - There was an error in the media server while trying to redirect the interaction to the default target.

The following are the queue asynchronous events:

Event	Attributes	Description
queue.cancelled		This event indicates that a target has been selected from another `<submit>` action request and that this `<submit>` action request has been cancelled.
	requestid	This is the submit request ID associated with the interaction.

Notes

Prior to version 8.1.200.27, the Interaction's age was ignored for Multimedia Interactions.

Classification Interface

This interface provides the ability to classify and screen interaction content to help the orchestration logic determine what the customer wants.

Object Model

category

This object represents the result for an interaction classified against a given category. The name of the object will be "category". This object is accessible either through the `classification.screen.done` or the `classification.classify.done` events and the categories property with those events. Another way is via the interaction object's categories property. This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
name	read only	string	none		This is the name of the screen rule.
id	read only	string	none		This is screen rule ID from UCS.
relevancy	read only	integer	none	1-100	This indicates the relevancy of the category with respect to the interaction that was classified.

screenrule

This object represents the result for an interaction screened against a given screen rule. The name of the object will be "screenrule". This object is accessible through the `classification.screen.done` event and the screenrule property of that event. This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
name	read only	string	none		This is the name of the screen rule
id	read only	string	none		This is screen rule ID from UCS.

Name	Access	Type	Default Value	Valid Values	Description
match	read only	boolean	none	true, false	This indicates whether the interaction has met the conditions of the screen rule.
value	read only	string	none		This is the interaction content that has met the screen rules.

Action Elements

<classify>

This action takes the content of an interaction and classifies it into categories. This action is equivalent to the IRD function block "Classify".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among

Name	Required	Type	Default Value	Valid Values	Description
					several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.
server	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the name of the classification server to be used for classifying the content of this interaction. If not supplied, the functional module will use the first available server. See SCXML Legal Data Values and Value Expressions for details.
interactionid	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the _genesys.ixn.interactions[x].g_uid of the interaction that is to be classified. There is a special value that can be returned: • ECMAScript **Null** means the functional module will not use an interaction for the request. SCXML Legal Data Values and Value Expressions for details.
root	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the overall classification

Name	Required	Type	Default Value	Valid Values	Description
					category that should be used for this action. SCXML Legal Data Values and Value Expressions for details.
confidence	false	value expression	75	Any value expression that returns a valid integer between 1 and 100	A value expression which returns the minimum relevancy each classification category must have in order for Classification Server to consider an interaction as belonging to that category. See SCXML Legal Data Values and Value Expressions for details.
fromudata	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the udata key name to use as the classified data source. If this attribute and the fromvar attribute are not supplied, then the data source will be Universal Contact Server's database. See SCXML Legal Data Values and Value Expressions
fromvar	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the ECMAScript variable name to be used as the classified data source. If this attribute and the fromudata attribute are not supplied then the data source will be Universal Contact Server's database. See SCXML Legal Data Values and Value Expressions for details.

Name	Required	Type	Default Value	Valid Values	Description
allcategories	false	boolean expression	true	Any expression which returns a boolean (true, false)	A boolean expression which indicates whether or not all categories are to be used for this action. This attribute is mutually exclusive with the categories and subcategories attributes. See SCXML Conditional Expressions for details.
categories	false	value expression	none	Any expression that results in a valid string	A value expression which returns a set of comma-separated category names (ids) that is to be used to classify this interaction. For example, categories="'productx, update'" This attribute is mutually exclusive with the `allcategories` attribute. See SCXML Legal Data Values and Value Expressions for details.
subcategories	false	NMTOKEN	no	no, direct, all	This identifies how the Classification Server should handle parent and child categories when classifying the interaction: • no - The Classification Server considers only the categories you selected in the tree and does not include any child categories. • direct - The

Name	Required	Type	Default Value	Valid Values	Description
					Classification Server considers only the direct children of the selected (parent) categories. • all - The Classification Server considers all children of the selected (parent) categories. This attribute is mutually exclusive with the allcategories attribute.
attach	false	boolean	true	true and false	This indicates whether the classification results should be attached to the interaction as udata properties.

The following is an example:

```
<state id="do_classification">
    <datamodel>
        <data id="reqid"/>
    </datamodel>
    <onentry>
        <classification:classify requestid="_data.reqid" root="businessxcats"/>
    </onentry>
    <transition event="classification.classify.done" target="statex"/>
    <transition event="error.classification.classify" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `classification.classify.done` - This event is sent when the request has been accepted by the system and the interaction has been classified.
- `error.classification.classify` - This event is sent when the request has failed for some reason.

<screen>

This action takes the content of an interaction and screens it using a set of rules. This action is equivalent to the IRD function blocks "Screen" and "Multiscreen".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.

Name	Required	Type	Default Value	Valid Values	Description
server	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the name of the classification server to be used for screening this interaction. If not supplied, the functional module will use the first available server. See SCXML Legal Data Values and Value Expressions for details.
interactionid	false	value expression	"0"	Any value expression that returns a valid string	A value expression which returns the `_genesys.FMname.interactions[x].` of the interaction that is to be screened. There is a special value that can be returned: • **"0"** means the functional module will use _genesys.FMname.interact as the related interaction. SCXML Legal Data Values and Value Expressions for details.
language	false	value expression	English (US)	Any expression that returns a string with one of the following values: English (US), Spanish, Mandarin, Cantonese, Vietnamese, French, French (Canada), German, Italian, Japanese, Korean, Russian	A value expression which returns a string specifying a language in which screening should be done. See SCXML Legal Data Values and Value Expressions for details.
fromudata	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the udata key name to use as the

Name	Required	Type	Default Value	Valid Values	Description
					screened data source. If this attribute and the fromvar attribute are not supplied, then the data source will be Universal Contact Server's database. See SCXML Legal Data Values and Value Expressions
fromvar	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the ECMAScript variable name to be used as the screened data source. If this attribute and the fromudata attribute are not supplied, then the data source will be Universal Contact Server's database. See SCXML Legal Data Values and Value Expressions for details.
allrules	false	boolean expression	true	Any expression which returns a boolean (true, false)	A boolean expression which returns whether or not all rules are to be used for this action. This attribute is mutually exclusive with the rules attribute. See SCXML Conditional Expressions for details.
rules	false	value expression	none	Any expression that results in a valid string	A value expression which returns a set of comma-separated rule names (ids) that is to be used to screen this interaction. For example, `rules="'rule1, rulen'"` This attribute is mutually exclusive with the allrules attributes. See SCXML Legal Data

Name	Required	Type	Default Value	Valid Values	Description
					Values and Value Expressions for details.
results	false	NMTOKEN	rules	rules, matchpairs, categories, all	This identifies what results the Classification Server should return after the interaction has been screened: • **rules** - Returns screening rule IDs when a match is found. • **matchpairs** - Returns the matched pairs of screening rule IDs and specific strings of words in the interaction content that matched the screening rules. • **categories** - Returns the classification categories associated with the screening rules. • **all** - Returns screening rule IDs, key-value pairs, and categories

Name	Required	Type	Default Value	Valid Values	Description
root	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the overall screening category which should be used for this action. This is attribute is mandatory when the results attribute is either "categories" or "all". SCXML Legal Data Values and Value Expressions for details.

The following is an example:

```
<state id="do_screening">
    <datamodel>
        <data id="reqid"/>
        <data id="rulesetA" expr="Rule1, Rule2, Rule3"/>
    </datamodel>
    <onentry>
        <classification:screen requestid="_data.reqid" rules="_data.rulesetA"
        results="all" />
    </onentry>
    <transition event="classification.screen.done" target="statex"/>
    <transition event="error.classification.screen" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `classification.screen.done` - This event is sent when the request has been accepted by the system and the interaction has been screened.
- `error.classification.screen` - This event is sent when the request has failed for some reason.

Events

Event	Attributes	Description
classification.classify.done		This event indicates the success of the request and that the interaction has been classified.
	requestid	This is the ID associated with the request.
	categories	This is an array of category objects which have met the criteria associated with this request and the interaction. The array is ordered from highest relevancy to lowest.
error.classification.classify		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or the interaction itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred: • Request Invalid
	description	This is a more detailed description of the error.
classification.screen.done		This event indicates the success of the request and that the interaction has been screened.
	requestid	This is the ID associated with the request.
	screenrule	This is an object which contains properties of screening rule what has met the criteria associated with this request and the interaction as well as collection of matched keys.
	categories	This is an array of category objects which have met the criteria associated with this request and the interaction. The array is ordered from highest relevancy to lowest.
error.classification.screen		This indicates that an abnormal condition

Event	Attributes	Description
		occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred: • Request Invalid
	description	This is a more detailed description of the error.

Core Extensions

Session Interface

Object Model

_genesys.session Object

Every SCXML session instance running in the orchestration platform will have an object with a set of common orchestration logic properties. These properties are maintained by the orchestration platform, but they can be set or updated by the orchestration logic itself. They are also used by the orchestration platform for orchestration logic reporting and management functionality. The name of the object will be "_genesys.session". This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
server	read only	genesys.session.server object	none		This is the Genesys server information on which this session is running.
tenant	read only	string	none		This is the name of the tenant that this session is associated with. It can be changed with the _genesys.session.setTenant() function.

_genesys.session.server Object

Every SCXML session instance running in the orchestration platform will have a global root object from which an application will have access to platform server information. This object is maintained by the orchestration platform. The name of the object will be "_genesys.session.server". This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
name	read only	string	none		This is the configuration layer application name for the active platform server running this session.

Name	Access	Type	Default Value	Valid Values	Description
cluster	read only	string	none		This is the configuration layer application name of the platform cluster (that is, the primary platform server) running this session.

_genesys.session.lookupseq ENUM Object

This represents the lookupsequence enumeration. This enumeration is maintained by the orchestration platform. This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
StartFromStrategy	Read only	Integer	None	-1	The lookup starts from the routing strategy
StartFromCDN	Read only	Integer	None	0	The lookup starts from the CDN
StartFromTserver	Read only	Integer	None	1	The lookup starts from the T-Server
StartFromTenant	Read only	Integer	None	2	The lookup starts from the Tenant
StartFromRouter	Read only	Integer	None	3	The lookup starts from the URS

_genesys.session.day ENUM Object

This represents the day enumeration. This enumeration is maintained by the orchestration platform. This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
Sunday	read only	integer	none	0	This represents Sunday.
Monday	read only	integer	none	1	This represent Monday.

Name	Access	Type	Default Value	Valid Values	Description
Tuesday	read only	integer	none	2	This represents Tuesday.
Wednesday	read only	integer	none	3	This represents Wednesday.
Thursday	read only	integer	none	4	This represents Thursday.
Friday	read only	integer	none	5	This represents Friday.
Saturday	read only	integer	none	6	This represents Saturday.

Functions

_genesys.session.dateInZone

This function returns the current date in the specified time zone. The results will be in the xml date datatype format (that is, yyyy-mm-dd). This can be compared with other variables that use the same time format. `date _genesys.session.dateInZone(tzone)` Parameters:

- **tzone:** STRING which can be a variable or a constant - This parameter is the name of a time zone configured in the configuration layer.

Returns: `date`: xml date datatype - This value represents the current date, based on the time zone specified. For example, "if (_genesys.dateInZone("EST") == "2009-01-28")".

_genesys.session.timeInZone

This function returns the current time in the specified time zone; that is, the number of minutes elapsed since the last midnight (00:00 AM) in the specified time zone. The results will be in the xml time datatype format (that is, hh:mm:ss or hh:mm). This can be compared with other variables that use the same time format. `time _genesys.session.timeInZone(tzone)` Parameters:

- **tzone:** STRING which can be a variable or a constant - This parameter is the name of a time zone configured in the configuration layer.

Returns: time: xml time datatype - This value represents the current time, based on the time zone specified. For example, "if (_genesys.timeInZone("EST") == "17:00:00")".

_genesys.session.dayInZone

This function returns the current day of the week in the specified time zone. The results will be a value from the _genesys.session.day enumeration. This can be compared with other objects that use the same enumeration object. `day _genesys.session.dayInZone(tzone)` Parameters:

- **tzone:** STRING which can be a variable or a constant - This parameter is the name of a time zone configured in the configuration layer.

Returns: day: genesys.session.day ENUM OBJECT which can be a variable or a constant - This value represents the current day of the week, based on the time zone specified. For example, "if (_genesys.session.dayInZone("PST") == 5)".

_genesys.session.isSpecialDay

This function checks to see if the current day and time is defined in the configuration layer as a special day. `value _genesys.session.isSpecialDay(stat_table, stat_day, zone, useTime)` Parameters:

- **stat_table:** STRING which can be a variable or a constant - This parameter is the stat table in the configuration layer which this function will check.
- **stat_day:** STRING which can be a variable or a constant - This parameter is optional. If it is specified, the platform inquires from the configuration layer whether the specified statistical day is configured for the specified statistical table and whether the current date meets the definition of the statistical day. If this parameter is not specified, the platform inquires from configuration layer whether the current date meets the definition of any of the statistical days configured for the specified statistical table.
- **zone:** STRING which can be a variable or a constant - This parameter is optional. It defines the timezone to be used to determine if the given day is a special one. If this parameter is not specified, then the current date and time are used in the current TimeZone. If this parameter is specified, then the specified time zone will be used to calculate the adjusted date and time.
- **useTime:** BOOLEAN which can be a variable or a constant - This parameter is optional. It specifies whether or not to use the time limits specified within the stat day definition in the configuration layer.

Returns: `value`: BOOLEAN - This indicates if the current day and time is special, based on configuration information.

_genesys.session.getConfigOption

This function allows the use of customized configuration options from the configuration layer and is obtained via URS - the user can configure any option name that is different from the standard options and then use its value in the session. In particular, you can specify any options you like in addition to the required ones and then give them meaning in the logic. This function retrieves the current value of any platform configuration option for use in the session. The search for the option starts with the object properties given by lookup sequence (the DN or resource that triggered the start of the session, the media server controlling this DN, the tenant to which they belong, or the orchestration platform). If the option is not found there, the search continues in the object properties corresponding to greater values of lookup sequence, in increasing order, until the option is found. The key should be located within a section called `__ROUTER__` of the corresponding objects from which the function will search. Failure to provide the key within such a section of the objects will result in an empty string being returned. Please refer to the Universal Routing Reference Manual for more details on setting URS Options and supported sections where options may be located from. `value _genesys.session.getConfigOption(ixnid, key, lookupseq)` Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.

- **key:** STRING which can be a variable or a constant - This parameter is mandatory. It is the key name of the configuration option in the configuration layer, it should be present under the required section of the objects, i.e. __ROUTER__
- **lookupseq:** genesys.session.lookupseq ENUM OBJECT which can be a variable or a constant - This parameter is mandatory. It defines the lookup sequence to use while searching the configuration layer..

Returns: `value`: STRING - This is the value of the configuration option key. The empty string is returned if the option is not found.

_genesys.session.getServerVersion

Starting with ORS 8.1.400.24, this function allows you to retrieve the Orchestration Server version.

- **Parameters:** None
- **Returns**: STRING (version of Orchestration Server)

_genesys.session.getValue

This function searches an object tree to find a specific property and returns the value of that property. This is needed for properties like _genesys.ixn.interactions[x].udata or _genesys.ixn.interactions[x].xdata. `object _genesys.session.getValue(object rObj, string key)` Parameters:

- **rObj:** OBJECT which can be a variable or a constant - This parameter is the object which is going to be searched.
- **key:** STRING which can be a variable or a constant - This parameter is the name of property to search for.

Returns: `value`: OBJECT - This is the value of the property. An empty object is returned if no property was found in the object tree.

_genesys.session.setOptions

This function is an override for platform-level configuration options. It enables the session to take control of certain options, instead of leaving them under the control of the platform or of functional modules. These changes only affect the current session and are not applied to the entire platform. **Note:** Using this function may negatively impact URS performance. `void _genesys.session.setOptions(ixnid, option, value)` Parameters:

- **ixnid:** STRING which can be a variable or a constant - This parameter is mandatory. It defines the ID of the interaction which should have this action applied.
- **option:** STRING which can be a variable or a constant - This parameter is the configuration layer option that is to be overridden. Previously, the options that could be overridden were as listed below. Starting with Release 8.1.400.17, this restriction is now removed.
  - request_timeout (all functional module requests)
  - null_value (not valid any more, because DB access will be through the `<fetch>` element)
  - default_object (Queue functional module)
  - use_ivr_info (Queue functional module)

    - default_destination (Queue functional module)
    - use_agentid (Queue functional module)
    - use_extrouter (Queue functional module)
    - use_extrouting_type (Queue functional module)
- **value:** STRING which can be a variable or a constant - This parameter is the value that is to be used as the override for the option.

Returns: `VOID`

_genesys.session.setTenant

This function overrides the tenant for this session. `void _genesys.session.setTenant(name)` Parameters:

- **name:** STRING which can be a variable or a constant - This parameter is the configuration layer name of the tenant to be set.

Returns: `VOID`

_genesys.session.getListItemValue

A developer can create string-related lists in the Configuration Layer. For example, these lists can be used to create lists of toll-free numbers instead of references for each individual 800 number in the logic. You can logically group numbers together and name the group. Then, when you need to add or edit numbers, the logic does not need changing; you just add to or edit the list. This function looks for an element item in the configured list and returns the value of its property key. `value _genesys.session.getListItemValue(list, item, key)` Parameters:

- **list:** STRING which can be a variable or a constant - This parameter is the name of the list in the configuration layer which this function will try and get the appropriate value for.
- **item:** STRING which can be a variable or a constant - This parameter is the name of the item in the list which this function will try and get the appropriate value for.
- **key:** STRING which can be a variable or a constant - This parameter is optional. It is the name of the key in the list which this function will try and get the appropriate value for. If this parameter is not specified, all properties of the found list elements are returned in as an OBJECT of key/value pairs: {Key1:value1, Key2:Value2, ...}.

Returns: `value`: STRING - This is the value of the key in the list that was found, or OBJECT - all key/value pairs if the key wasn't specified. If the item or key is not found, this function returns an empty string. If list object itself is not found the function returns error (i.e. raises exception).

Starting with ORS 8.1.400.49, if option `functions-by-urs` is `false`, an asterisk ('*') can be specified as the value of item in the second parameter of the `_genesys.session.getListItemValue(list, item, key)` function. The key parameter is then mandatory and cannot be an asterisk. In this case, ORS performs a lookup for the specified key among all items. If only one key is found, the function returns a string value that corresponds to the key. If the key is found in several items, then the function returns object: `{Item1:Value1, Item2:Value2…}`

_genesys.session.listLookupValue

This function checks whether a List object in the configuration layer contains a particular element item.

`value _genesys.session.listLookupValue(list, item)` Parameters:

- **list:** STRING which can be a variable or a constant - This parameter is the name of the list in the configuration layer which will be searched to determine whether it contains the item.
- **item:** STRING which can be a variable or a constant - This parameter is the name of the item that will be searched for in the list.

Returns: `value`: BOOLEAN - This return value indicates whether the the item is part of the list.

_genesys.session.getServerVersion

Starting from ORS 8.1.400.22, this function returns version of Orchestration Server. `value _genesys.session.getServerVersion()`

Parameters: None

Returns: `value`: STRING - This is the version of Orchestration Server.

Action Elements

This covers action elements that are related to SCXML sessions, but are Genesys-specific. The namespace for these session-related actions is www.genesyslab.com/modules/session .

<fetch>

This action element fetches business content or data from an application server and is an enabler for Orchestration Server Integration within a customer's environment. The content could be generated by actual "business logic" running on the application server or it could just be a static content file on the application server, which could be updated (even manually) as required to allow things to be dynamic. The business content and associated logic itself will be created based on the programming technology of the application server it is going to run on. So the application server-specific development tools will be used to create this content and associated logic. This element is used within executable content processing. This business content and associated logic will be deployed and executed on an application server. The orchestration platform will support both .NET and J2EE application servers. The form of the returned business content will be JSON.

There is no explicit context or state shared between the orchestration platform and the application server. All context or state that is needed by the business content and logic must be sent via this action element. If the needed context or state is not totally known at the time of invocation, then the developer can use the QuerySessionData web services within their business content-fetching logic to get the current orchestration logic context for the given orchestration logic session. This provides a more flexible and dynamic mechanism. It also allows the developer to optimize what context or state is needed for each business content-fetching logic "application". This element may cover web service invocations in the future. In the meantime, the customer can use business content-fetching logic as a proxy for executing web services. This is also the way to invoke both database-related actions and rules system-related actions.

In addition, this action will support ESP-based requests from Genesys Interaction Server through the ESP (External Service Protocol) protocol and access URS REST API via direct ORS-URS connection. For HTTP and HTTPS, basic authentication is supported if the username and or password is provided in the request. This will result in the request being submitted to the application server with the Authorization HTTP header element added to the message. In addition to this, for both HTTP and HTTPS additional headers may be provided by passing an ECMAScript object into the request.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
attach_ixn_data	false	Boolean	true/false	Any valid location expression which represents a string	Introduced in 8.1.400.48. Use to enable/disable attachment of interaction properties. ORS will ignore this attribute if “method” attribute is not “’esp’”. Default value of that attribute is true. If true, ORS will populate hr UserData in the ESP request with the following interaction properties: InteractionId, ParentId, TenantId, MediaType, InteractionType, InteractionSubtype, InteractionState, IsOnline, IsLocked, Queue, Workbin, WorkbinAgentId, WorkbinAgentGroupId, WorkbinPlaceId, WorkbinPlaceGroupId, SubmittedBy, ReceivedAt, SubmittedAt, DeliveredAt, SubmittedToRouterAt, PlacedInQueueAt, MovedToQueueAt, AbandonedAt, SubmitSeq, PlaceInQueueSeq, HeldAt, IsHeld, AssignedAt, CompletedAt, AssignedTo.
requestid	false	location expression	none	Any valid location expression which	This is the location for the request ID

Name	Required	Type	Default Value	Valid Values	Description
				represents a string	that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
srcexpr	true	value expression	none	Any value expression that returns a valid string URI of the following types: • HTTP • HTTPS • File • gesp	This value expression will be evaluated at the time that the fetch element is executed to produce the URI to pass to the application server. The URI schemes supported are HTTP, HTTPS and File. **Note:** when the scheme is 'gesp', the URI will have a specific format. See the ESP based `<fetch>` actions section for details. **Note:** when the method is 'urs', the URI will have a specific format. See SCXML Legal Data Values and Value

Name	Required	Type	Default Value	Valid Values	Description
					Expressions for details.
type	false	value expression	application/json	application/xml, application/json, text/plain	This value expression returns a character string that specifies the type of the fetched content. Values defined by the specification are: • application/xml - This specifies that the document being fetched must be an XML document for a given namespace. • application/json - This specifies that the fetched content must be JSON format. This is the default. • text/plain - This specifies that the fetched content must be plain text in format. If method attribute is "esp" or "urs", this attribute is ignored. The type attribute in the issued HTTP request will be passed to the application server as the Accept

Name	Required	Type	Default Value	Valid Values	Description
					header. See SCXML Legal Data Values and Value Expressions for details.
method	false	value expression	get	get post esp urs put delete	A value expression which returns a character string that indicates the HTTP method to use. See SCXML Legal Data Values and Value Expressions for details. Values defined by the specification are: • get - This indicates that the "GET" method must be used to fetch the URL. • post - This indicates that the "POST" method must be used while submitting the URL to the web server. • esp - This indicates that the ixn-server ESP protocol is to be used. • urs - This that URS REST API will be used via direct connection • put - This

Name	Required	Type	Default Value	Valid Values	Description
					indicates that the "PUT" method must be used while submitting the URL to the web server. • delete - This indicates that the "DELETE" method must be used while submitting the URL to the web server.
timeout	false	value expression	0	A value expression which returns an integer	A value expression which returns an integer that represents the number of seconds to wait. See SCXML Legal Data Values and Value Expressions for details. The integer returned must be interpreted as a time interval. This interval begins when `<fetch>` is executed. A failed and timed out fetch must return the error.session.fetch event.
maxage	false	value expression		A value expression which returns a valid integer for the HTTP 1.1 request RFC 2616	The integer returned must be interpreted as a time interval. This indicates that the logic is willing to use content whose age must be no greater than the specified time in seconds (compare with 'max-age' in HTTP 1.1 RFC

Name	Required	Type	Default Value	Valid Values	Description
					2616). The logic is not willing to use stale content, unless maxstale is also provided. If method attribute is "esp" or "urs", this attribute is ignored.
maxstale	false	value expression		A value expression which returns a valid integer for the HTTP 1.1 request RFC 2616	The integer in string form returned must be interpreted as a time interval. This indicates that the logic is willing to use content that has exceeded its expiration time (cf. 'max-age' in HTTP 1.1 RFC 2616). If maxstale is assigned a value, then the logic is willing to accept content that has exceeded its expiration time by no more than the specified number of seconds. If method attribute is "esp" or "urs", this attribute is ignored.
username (since 8.1.1)	false	value expression	none	Any expression that results in a valid string value	This value expression returns a character string that represents the username to be used as apart of HTTP Basic Authentication as defined by HTTP 1.1 [See RFC 2616]. If method attribute is "urs", this attribute is ignored.
password (since 8.1.1)	false	value expression	none	Any expression that results in a valid string value	This value expression returns a character string that represents the password to be used as apart of HTTP Basic Authentication as defined by HTTP 1.1 [See RFC

Name	Required	Type	Default Value	Valid Values	Description
					2616]. If method attribute is "urs", this attribute is ignored.
enctype	false	value expression	application/x-www-form-urlencoded	application/x-www-form-urlencoded, application/json	This value expression returns a character string that specifies the type of encoding to be used for the content of the POST/PUT message. Values defined by the specification are: • application/x-www-form-urlencoded - This specifies that the message content must be encoded in URL encoded form. This is the default. • application/json - This specifies that the message content must be encoded in JSON format. If method attribute is "esp" or "urs", this attribute is ignored. The content type returned by the application server response will be checked against the enctype attribute. If it is different, an error.session.fetch

Name	Required	Type	Default Value	Valid Values	Description
					event will be raised - the exception is the text/plain value case, in which case any type of returned value is accepted.The returned body is provided "as-is" in the content of the session.fetch.done event. The application logic is supposed to use the JSON functions to convert it into appropriate values. See SCXML Legal Data Values and Value Expressions for details.
gdelivery	false	value expression	false	A value expression which returns a boolean value (true or false)	A value expression which returns a boolean value that indicates whether the platform is to guarantee the execution of the `<fetch>` action. This does not guarantee that the action associated with the srcexpr value has been carried out successfully. It just guarantees that the HTTP request gets to the defined destination (srcexpr value) and that a response (positive or negative) is returned. If method attribute is "urs", this attribute is ignored. See SCXML Legal Data Values and Value Expressions for details.
gd_retries	false	value expression	0	A value expression which returns a valid integer	A value expression which returns an integer that indicates the number of times the platform should try to successfully deliver the associated

Name	Required	Type	Default Value	Valid Values	Description
					HTTP request to the defined destination (srcexpr value). This attribute is ignored if the gdelivery attribute value is false. If the gdelivery attribute value is true and the gd_retries value is 0, the platform will try to delivery the associated HTTP request indefinitely. If method attribute is "urs", this attribute is ignored. See SCXML Legal Data Values and Value Expressions for details.
gd_retry_interval	false	value expression	0	A value expression which returns a valid integer	A value expression which returns an integer that represents the number of seconds to wait. The integer returned must be interpreted as a time interval. This interval is the time to wait between retries. This interval begins after a failed retry. This attribute is ignored if the gdelivery attribute value is false. If method attribute is "urs", this attribute is ignored. See SCXML Legal Data Values and Value Expressions for details.
headers	false	value expression	none	Any valid ECMAScriipt object	A value expression which returns an ECMAScript object. Each property name within the object will be interpreted as a separate HTTP header. Its value will be obtained using toString()

Name	Required	Type	Default Value	Valid Values	Description
					and appended after the property name followed by a ":" character. No checking or validation will be performed on the properties and or values provided within the object. This will not override any headers automatically added by <fetch> such as Cache-Control and is provided as a means to provide the ability to add customer header information. If method attribute is "urs", this attribute is ignored. See SCXML Legal Data Values and Value Expressions for details.

Important Note: Any evaluated attribute, if specified, must evaluate to a valid value. Otherwise, an `error.script` event will be generated. This also applies to attributes that are ignored under specific conditions. The mapping of this action to the underlying HTTP request and response is described in the Mapping of the SCXML and Functional Module Elements to the HTTP Messages section.

The following are examples of the `<session:fetch>` action.

The example below demonstrate how to use <fetch> with method="'urs'". If you need to call a function from the URS REST API, the example demonstrates how to specify function name, how to pass parameters, and how to retrieve the returning object. The example is applicable for any function, not only FindConfigObject as is used below.

```
<state id="FindPersonByEmployeeID">
<datamodel>
   <data id="reqid" />
</datamodel>

<onentry>
<script>
                var s_URI = 'urs/call/@' + system.InteractionID + '/func';
                var message = [3, "employeeid:EID_1000"];
</script>
<session:fetch requestid="_data.reqid" srcexpr="s_URI" method="'urs'">
                <param name= "name" expr="'FindConfigObject'" />
                <param name= "params" expr="uneval(message)" />
</session:fetch>
</onentry>
<transition event="session.fetch.done" target="statex">
   <script>
     var PersonObject = eval("(" + _event.data + ")");
   </script>
   <assign location="_interactionID"
expr="eventDataObject.envelope.Parameters.InteractionId"/>
</transition>
<transition event="error.session.fetch" target="statey" />
</state>
```

Another example:

```
<state id="get_business_data_using_param_child">
<datamodel>
 <data id="reqid"/>
 <data id="customervalue"/>
</datamodel>
<onentry>
 <session:fetch requestid="_data.reqid" srcexpr="'www.joes.com\getbusinessdata'" timeout="30">
   <param name="customerID" expr="_cv.customerid"/>
 </session:fetch>
</onentry>
<transition event="session.fetch.done" target="statex">
 <assign location="customervalue" expr="_event.data.cvalue"/>
</transition>
<transition event="error.session.fetch" target="statey"/>
</state>
<state id="get_business_data_using_content_child">
<datamodel>
 <data id="reqid"/>
 <data id="complexobject"/>
</datamodel>
<onentry>
 <session:fetch requestid="_data.reqid" srcexpr="'www.joes.com\getbusinessdata'" timeout="30">
   <content _expr="_data.complexobject"/>
 </session:fetch>
</onentry>
<transition event="session.fetch.done" target="statex">
 <assign location="customervalue" expr="_event.data.cvalue"/>
</transition>
<transition event="error.session.fetch" target="statey"/>
</state>
<state id="get_business_data_using_basic_auth_and_headers">
<datamodel>
 <data id="reqid"/>
 <data id="customervalue"/>
</datamodel>
<onentry>
 <script>
   var myheaders = new Objects();
   myheaders["If-Modified-Since"] = "Sat, 1 Jan 2011 20:00:00 GMT";
   myheaders["X-CUSTOM-HEADER"] = "Custom header information";
 </script>
 <session:fetch requestid="_data.reqid" srcexpr="'www.joes.com\getbusinessdata'" timeout="30"
username="'bob'" password="'mysecret'" headers="myheaders" >
   <param name="customerID" expr="_cv.customerid"/>
 </session:fetch>
</onentry>
<transition event="session.fetch.done" target="statex">
 <assign location="customervalue" expr="_event.data.cvalue"/>
</transition>
<transition event="error.session.fetch" target="statey"/>
</state>
```

Children

- `<param>` Occurs 0 to N. See SCXML `<param>` for details. This element is mutually exclusive with the `<content>` element. These parameters will be submitted differently, depending on the type of message used:
  - **HTTP GET/DELETE Message** - The `<param>` elements yield a URL parameter list (name1= value1&name2=value2...) at the end of the path element. Note that if the srcexpr attribute evaluates to a URL with URL parameters, the `<param>` element parameters will be concatenated to the end of the URL component. If the `<param>` element value is a complex object, the value is the result of evaluating the ECMAScript toString() function of that object, which is usually the string "[object object]". As a result, instead of submitting objects directly, the application developer must explicitly submit the properties of an object, for example, "_genesys.session.server.name". The following is the mapping of the `<param>` element attributes to the URL parameter list format:
    - name - The "name" attribute of the `<param>` element will be submitted with the given parameter value as its key.
    - value - The current value associated with this `<param>` element "expr" attribute:
      - Simple types (string, integer, boolean, or decimal) will be converted into strings.
      - ECMAScript objects will be converted into the string "[object object]". Complex objects should not be used.
  - **HTTP POST/PUT Message** - The `<param>` elements yield different formats depending on the `<fetch>` enctype attribute value. Regardless, the results will be put into the body element of the POST/PUT message:
    - **application/x-www-form-urlcoded** - The `<param>` elements for this encoding format will be transformed into a URL parameter list (name1=value1&name2=value2...) which will be inserted into the body element of the message. If the `<param>` element value is a complex object, the value is the result of evaluating the ECMAScript toString() function of that object, which is usually the string "[object object]". As a result, instead of submitting objects directly, the application developer must explicitly submit the properties of an object, for example, "_genesys.session.server.name". The following is the mapping of the `<param>` element attributes to the URL parameter list format:
      - name - The "name" attribute of the `<param>` element will be submitted with the given parameter value as its key.
      - value - The current value associated with this `<param>` element "expr" attribute:
        - Simple types (string, integer, boolean, or decimal) will be converted into strings.
        - ECMAScript objects will be converted into the string "[object object]". Complex objects should not be used.
    - **application/json** - The `<param>` elements for this encoding format will be a JSON-formatted string which will be inserted into the body element of the message. Each `<param>` element will be a top-level attribute in the format. For example, `<param name="a" expr="value1"/>` `<param name="b" expr="complexb"/>` `<!- complexb has three properties e,f,g ->` will result in the following JSON-formatted string - {"a":value1, "b":{"e":88, "f":"john", "g":22}}.
  - **ESP Message** - The `<param>` element can only be used to pass request parameters on the ESP request message. If the ESP request message requires the passing of user-data parameters as well, then the `<content>` element MUST be used instead. The `<param>` elements will put in the Interaction Server TKVList format. The `<fetch>` enctype attribute value is ignored.
  - **urs Message** - the same as for HTTP GET/DELETE

- `<content>` Occurs 0 to 1. See SCXML `<content>` for details. This element is mutually exclusive with the `<param>` element. The content defined in this element will match the value of the `<fetch>` element enctype attribute:
  - application/x-www-form-urlencoded - The _expr attribute cannot be used and the content of this element will be sent as is.
  - application/json - The _expr attribute must be specified and must evaluate to an ECMAScript object. The object will be sent as is.

Summary of URL and JSON encoding

The following table is a summary of the rules described above.

	URL Encoded		**JSON Encoded**	
<param>	<content>	<param>	<content>	
GET, DELETE	Name and expression attributes of each <param> element (*name* and *expr*, correspondingly) will be evaluated, URL-encoded, and combined into a standard name-value sequence (n1=v1&n2=v2&...). Duplicate parameter names are acceptable. The following rules will be used when URL-encoding the expression attribute: • String, number, true, false, and null - as usual. • Any ECMAScript object will be encoded to the following: `%5Bobject%20Object%5D` • Array will be encoded as a comma-separated string of values. For example, [1,2,A] will be encoded as the following: `1%2C2%2C%5Bobject%20Object%5D`	Content of the <content> element will be URL-encoded without prior evaluation. This element cannot be used together with <param>.	Not supported.	Not supported.
POST, PUT			Name and expression attributes of each <param> element (*name* and *expr*, correspondingly) will be evaluated, combined into a single object and then converted into the JSON string. Duplicate parameter names are acceptable, however, remember that duplicate property names in JSON string will be eliminated during evaluation. For example, the ECMAScript expression: `eval('({"p1":1,"p1":2})');` will return the following object: `{"p1": 2}`	Expression attribute (*_expr*) of the <content> element will be evaluated and converted into the JSON string. This element cannot be used together with <param>.

Events

The following events can be generated as part of this action:

- session.fetch.done
- error.session.fetch

As of 8.1.200.50, upon successful execution of <session:fetch>, the event `session.fetch.done` is generated. It is possible to retrieve the response headers of the HTTP request from this event using: `_event.data.headers`. The headers are provided as key-value pairs.

> **Important**
> Currently, the <fetch> element in ORS does not extract the response payload for a non-200 OK response for HTTP requests. That is, the HTTP response body is not extracted in case of a non-200 OK response.

ESP-Based `<fetch>` Actions

For backwards compatibility purposes the platform and the `<fetch>` element will support the invocation of External Service Protocol (ESP) requests and responses. In order to use the `<fetch>` element for ESP-related requests, the developer needs to do the following:
`gesp:[<applname>]|[\<type>\]<service>\[<method>]` The following are the meanings of the different elements of the format:
For example,

```
MyEmailServer\CFGEmailServer\EMail\CreateEmailOut
\CFGContactServer\Contact\Update
```

- Specify a value of 'esp" for the method attribute.
- Construct the gesp URI with the following format for the srcexpr attribute:
  - - **"applname"** is the 3rd party application (connected to Interaction Server) that is to be used to process this request.
    - **"type"** is the 3rd party application type that is to be used to process this request. (optional)
    - **"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. (optional)

> **Important**
> If the type is specified as CFGInteractionServer in the gesp URI, ORS ignores "applname". Instead, it sends the ESP request to the Interaction Server that handles multimedia interactions and is attached to the current session. If there is no such

interaction (for example, if we have a voice call-initiated session), the ESP request will be sent to a random Interaction Server among those connected to ORS.

- The only way to pass request parameters and interaction user data on an ESP-based `<fetch>` action is with the `<content>` element. You use the _expr attribute with an ECMAScript object which contains properties called "params" and "udata". The one with the name "udata" will be an ECMAScript object which contains the user-data parameters that you want to pass with the action. The other properties in the main ECMAScript object will be request parameters. All of these parameters will be in the Interaction Server TKVList format.
- The `<fetch>` type, enctype, maxage, and maxstale attributes are ignored.
- The response data content will be return as a JSON string in the session.fetch.done event.

The following is an example of the `<session:fetch>` action:

```
<state id="updateContact">
 <datamodel>
   <data id="reqid" />
 </datamodel>
 <onentry>
 <script>
   var updateContactRequestContent = {
    params: {
     UseDataFromParameters: false
     },
    udata: {
     TenantId: 101,
     ContactId: "GK4MW583K80DTE04",
     FirstName: "James",
     LastName: "Johnson"
     }
   };
 </script>
 <session:fetch method="'esp'" requestid="_data.reqid"
 srcexpr="'ContactServer_801_04\\CFGContactServer\\Contact\\Update'">
   <content _expr="updateContactRequestContent" />
 </session:fetch>
 </onentry>
 <transition event="session.fetch.done" target="statex">
   <script>
     var eventDataObject = eval("(" + _event.data + ")");
   </script>
   <assign location="_interactionID"
expr="eventDataObject.envelope.Parameters.InteractionId"/>
 </transition>
 <transition event="error.session.fetch" target="statey" />
</state>
```

For a successful ESP call the returned response will use the following conversion logic to determine how the JSON object is structured.

- The ESP JSON response will contain an `envelope` and `user_data` only if the corresponding sections are returned with the data section of the ESP response.
- The ESP response containing the ESP `envelope` will contain `Service` and `Method` properties together with the associated envelope `Parameters` that will contain the key name value properties.

- The ESP response containing the ESP `user_data` will contain the key name value properties if provided in the ESP response.
- For both the envelope `Parameters` and also for the `user data` if any key names are duplicated in the ESP response then these will be converted to a JSON array of values. For entries that may have incompatible types but share the same name then such values will be incorporated into the same named array but provided as an object, rather than a value.

The following is an example of the above ESP-to-JSON conversion logic operating on the following ESP response.

```
06:22:54.082 'external_srvice_response' (501) message:
 attr_ref_id [int] = 294014
 attr_envelope [list, size (unpacked)=604] =
 'Service' [str] = "Contact"
 'Method' [str] = "Identify"
 'Parameters' [list] = (size=228)
 'ContactCreated' [str] = "false"
 'ContactIdList' [str] = "0005Ua6CJC69002J"
 'ContactIdList' [str] = "0005Ua6CJC69002N"
 'ContactIdList' [str] = "0005Ub6CJC6H0001"
 'ContactIdList' [str] = "0005Ub6CJC6H0004"
 'ContactIdList' [str] = "0005Ub6CJC6H0007"
 'NumberOfContactsFound' [int] = 5
```

The following would be the JSON representation of the successful ESP call.

```
{
 "envelope": {
 "Service": "Contact",
 "Method": "Identify",
 "Parameters": {
 "ContactCreated": "false",
 "ContactIdList": [
 "0005Ua6CJC69002J",
 "0005Ua6CJC69002N",
 "0005Ub6CJC6H0001",
 "0005Ub6CJC6H0004",
 "0005Ub6CJC6H0007"],
 "NumberOfContactsFound": 5
 }
 }
}
```

For ESP calls that return user data in addition to an enevelope, the following would be expected to be represented.

```
{
 "envelope": {
 "Service": "Contact",
 "Method": "Identify",
 "Parameters": {
 "ContactCreated": "false",
 "ContactIdList": "0005Ub6CJC6H0001",
 "NumberOfContactsFound": 1
 }
 },
 "user_data": {
 "FirstName": "James",
 "ContactId": "0005Ub6CJC6H0001",
 "LastName": "Johnson"
 }
}
```

Guaranteed Delivery of the `<fetch>` Action

When a web service request is made to the external service using `<fetch>`, the orchestration platform guarantees delivery of such a request. That is, the orchestration platform handles situations in which normal web service requests cannot be fulfilled. Here are some situations in which a web service request cannot be fulfilled:

- Cannot establish a connection to a given URI
- Received a redirect response when making a request to a given URI
- Received an error response when making a request to a given URI
- No response received to the web service request during timeout

To handle these situations, the orchestration platform would queue web service requests, and retry making the request within the configured timeout period. The orchestration platform would queue web service requests on the basis of their URIs. Note that this functionality addresses both cases of:

- Temporary unavailability of the external service
- Unavailability of one of the nodes in the load-balanced external service

Important Note: If the session exits a state that has an outstanding `<fetch>` action, then the outstanding `<fetch>` action will be terminated. Also if an invoked session with an outstanding `<fetch>` action terminates, then the outstanding `<fetch>` action will be terminated.

Mapping of the SCXML and Functional Module Elements to the HTTP Messages

The following sections cover the mapping of the SCXML and functional module element's attributes into the corresponding HTTP message elements.

Get/Delete message

Here is an example of the `<fetch>` action including basic authentication and optional headers:

```
<script>
  var myheaders = new Object();
  myheaders["If-Modified-Since"] = "Sat, 1 Jan 2011 20:00:00 GMT";
  myheaders["X-CUSTOM-HEADER"] = "Custom header information";
</script>
<session:fetch requestid="_data.reqid"
               srcexpr="'http://www.business1.com/data2/content'" type="'text/plain'"
               method="'get or delete'" timeout="100" maxstale="10" maxage="20"
               username="'open'" password="'sesame'" headers="myheaders">
  <param name="param1" expr="'value1'"/>
  <param name="p2" expr="'v2'"/>
</session:fetch>
```

Here is how it maps to an HTTP GET/DELETE message:

```
GET/DELETE /data2/content?param1=value1&p2=v2 HTTP/1.1
Host: www.business1.com
Cache-Control: max-age=10, max-stale=10
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
If-Modified-Since: Sat, 1 Jan 2011 20:00:00 GMT
X-CUSTOM-HEADER: Custom header information
...
```

Mapping Summary:

- The results of evaluating the `srcexpr` attribute yields the host and path elements of the HTTP message.
- The result of evaluating the child `<param>` elements yields the URL parameter list at the end of the path. Note that if the `srcexpr` attribute evaluates to a URL with URL parameters, we would concatenate the `<param>` element parameters to them. If the `<param>` element value is a complex object, the value is the result of evaluating the `toString()` function of that object, which is usually `[object object]`.
- The result of evaluating the `maxage` attribute yields the `Cache-Control` header with the `max-age` directive.
- The result of evaluating the `maxstale` attribute yields the `Cache-Control` header with the `max-stale` directive.
- The result of evaluating the `username` and `password` yields the addition of the Authorization header with the type specified as `basic` and the `username` and `password` base64 encoded.
- The result of providing `header` yields a `header` element for each of the items provided within the supplied object. No validation will occur on this and the headers will be appended "as-is".
- The result of evaluating the `type` attribute yields the `Accept` header. Also the data type returned by the application server in the HTTP response will be checked against the type value. If it is different, an `error.session.fetch` will be raised - an exception that is `text-plain`, which means any type of returned value is accepted.
- There is no HTTP body for the GET request.

Note: The `Basic authentication` and `optional` headers will operate exactly the same for `POST` or `PUT` types.

POST/PUT Message

Here is an example of the `<fetch>` action with enctype = application/x-www-form-urlcoded:

```
<session:fetch requestid="_data.reqid"
               srcexpr="'http://www.business1.com/data2/content'" type="'text/plain'"
               method="'post or put'" timeout="100" maxstale="10"
               maxage="20" enctype="'application/x-www-form-urlencoded'">
  <param name="param1" expr="'value1'"/>
  <param name="p2" expr="'v2'"/>
  <param name="p3" expr="v3"/>
</session:fetch>
```

Note: v3 is an object with two properties: "a" and "b". v3.a = 4 and v3.b = 5. Here is how it maps to an HTTP POST/PUT message:

```
POST/PUT /data2/content HTTP/1.1
Host: www.business1.com
Cache-Control: max-age=10, max-stale=10
Content-Type=application/x-www-form-urlencoded
```

```
Content-Length=xx
param1=value1&p2=v2&p3=[object object]
...
```

Here is an example of the `<fetch>` action with enctype = application/json:

```
<session:fetch requestid="_data.reqid"
               srcexpr="'http://www.business1.com/data2/content'" type="'text/plain'"
               method="'post or put'" timeout="100" maxstale="10"
               maxage="20" enctype="'application/json'">
  <param name="param1" expr="'value1'"/>
  <parm name="p2" expr="'v2'"/>
  <parm name="p3" expr="v3"/>
</session:fetch>
```

Note: v3 is an object with two properties "a" and "b". v3.a = 4 and v3.b = 5. Here is how it maps to an HTTP POST/PUT message:

```
POST/PUT /data2/content HTTP/1.1
Host: www.business1.com
Cache-Control: max-age=10, max-stale=10
Content-Type=application/json
Content-Length=xx
{"param1":"value1","p2":"v2","p3":{"a":4,"b":5}}
...
```

Mapping Summary:

- The results of evaluating the `srcexpr` attribute yields the host and path elements of the HTTP message.
- The results of the `<param>` elements depend on the `enctype` attribute. **Note:** the `Content-Length` header value will be set to the total length of the resulting body element.
- `application/x-www-form-urlcoded` - The result of evaluating the `<param>` elements yields the body element in the format p1=v1&p2=v2&p3=v3`..., where p1, p2, p3,...` are the names in the `<param>` elements, and `v1`, `v2`, and `v3` are values that result from evaluating the corresponding `expr` attributes. If one of the values is not a simple type, we would put the result of the "`toString()`" function in the body, as in the `GET` case.
- `application/json` - The result of evaluating the `<param>` elements yields the body element formatted in JSON format, where each `<parameter>` name should appear as a top-level attribute.
- The result of evaluating the `maxage` attribute yields the `Cache-Control` header with the `max-age` directive.
- The result of evaluating the `maxstale` attribute yields the `Cache-Control` header with the `max-stale` directive.
- The result of evaluating the `enctype` attribute yields the `Content-Type` header value.
- The result of evaluating the `type` attribute yields the `Accept` header. Also the data type returned by the application server in the HTTP response will be checked against the type value. If it is different, an `error.session.fetch` will be raised - this exception is `text-plain`, which means any type of returned value is accepted.

<start>

This starts an independent SCXML document and session and runs completely independently of the starting (that is, parent) session. If the starting session ends, the started session does not, and vice versa. Session content is not shared between the sessions. Any session or 3rd party application can terminate (for example, `<terminate>`) this started session, as long as they have the session ID. When this session terminates, a done event will be fired by the orchestration platform for all interested parties (other sessions, and so on).

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
sessionid	false	location expression	none	Any value location that represents a valid string field	This is the location for the session ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the newly created session. This ID is immediately populated by the platform and can be used on subsequent elements (`<send>`, for example).
src	true	value expression	none	Any value expression that returns a valid URI	This value expression returns a character string that represents the URI of the SCXML document. The URI schemes supported are HTTP, HTTPS and File. See SCXML Legal Data Values and Value Expressions for details. Starting with 8.1.400.09, Orchestration Server provides the ability to use the Enhanced Routing Script object when starting a new session by the `<session:start>` action. The script

Name	Required	Type	Default Value	Valid Values	Description
					name in the `script:ScriptName` format can be defined as a value of the `src` attribute of the `<session:start>` action element. When the `script:` notation is used, the URL of the SCXML strategy is taken from the `Application` section of the corresponding Enhanced Routing Script. Example: `<session:start src="’script:Script1’" sessionid="newid"/>`
idealtime	false	value expression	none	Any expression that results in a valid integer for the dateTime value	This value expression returns a dateTime value which will represent the date and time that this session is to be started. This value should be the time as returned by the ECMAScript Date(...).getTime() function, which is given in the number of milliseconds since 00:00:00 UTC on January 1, 1970. See SCXML Legal Data Values and Value Expressions for details.
prewindow	false	value expression	none	Any expression that results in a valid integer for the duration value	This value expression returns a duration value which will represent the time window prior to the ideal time for which the session could be started. For details on the duration type, see the duration datatype . See SCXML Legal Data Values and Value Expressions for details.

Name	Required	Type	Default Value	Valid Values	Description
postwindow	false	value expression	none	Any expression that results in a valid integer for the duration value	This value expression returns a duration value which will represent the time window after the ideal time for which the session could be started. For details on the duration type, see the duration datatype . See SCXML Legal Data Values and Value Expressions for details.

The following is an example:

```
<state id="Starting_a_new_session">
 <datamodel>
   <data id="newsession"/>
 </datamodel>
 <onentry>
    <session:start src="'www.genesyslab.com\session\orchapp1'" sessionid="_data.newsessid" />
 </onentry>
 <transition event="session.start.done" target="statex">
     <send event="'start.event'" target="_data.newsessid"/>
 </transition>
 <transition event="error.session.start" target="statey"/>
</state>
```

Children

- `<param>` Occurs 0 to N - This contains data to be passed to the newly created session. The use of this element will follow the same rules as the SCXML `<invoke>` element definition.

Events

The following events can be generated as part of this action:

- session.start.done
- error.session.start
- session.restored

<updatestart>

This action updates the starting time of the SCXML session. It can only be used after the session is started using the idealtime attribute and if the requested session has not been started yet.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
sessionid	true	value expression	none	Any value expression which returns a valid string.	A value expression which returns the session ID to be updated. See SCXML Legal Data Values and Value Expressions for details.
idealtime	false	value expression	none	Any expression that results in a valid integer for the dateTime value	This value expression returns a dateTime value which will represent the updated date and time that this session is to be started. This value should be the time as returned by the ECMAScript Date(...).getTime() function, which is given in the number of milliseconds since 00:00:00 UTC on January 1, 1970. See SCXML Legal Data Values and Value Expressions for details.
prewindow	false	value expression	none	Any expression that results in a valid integer for the duration value	This value expression returns a duration value which will represent the updated time window prior to the ideal time for which the session could be started. For details on the duration type, see the duration datatype. See SCXML Legal Data Values and Value Expressions for details.
postwindow	false	value expression	none	Any expression that results in a valid integer for the duration value	This value expression returns a duration value which will represent the updated time window after the

Name	Required	Type	Default Value	Valid Values	Description
					ideal time for which the session could be started. For details on the duration type, see the duration datatype . See SCXML Legal Data Values and Value Expressions for details.

Children

None

Events

The following events can be generated as part of this action:

- session.updatestart.done
- error.session.updatestart

<terminate>

This action terminates an SCXML session from an unrelated SCXML session. As a result of termination, the orchestration platform sends the done event to the invoking SCXML session (if it is still running). It will also be used to cancel a scheduled session.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
sessionid	true	value expression	none	Any value expression which returns a valid string.	A value expression which returns the session ID to be terminated. See SCXML Legal Data Values and Value Expressions for details.

Children

None

Events

The following events can be generated as part of this action:

- session.terminate.done
- error.session.terminate

<cancel>

This action terminates a pending fetch action request `<session:fetch>`. This is used to allow the application to ensure that any guaranteed delivery fetch requests are terminated. This action should be put in the `<onexit>` element where these types of fetch actions are invoked.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	true	value expression	none	Any valid value expression which returns a valid string.	This is the request ID of the outstanding `<fetch>` action.

Children

None

Events

The following events can be generated as part of this action:

- session.cancel.done
- error.session.cancel

Events

The event namespace convention is session.xxxx The following are the session action result events:

Event	Attributes	Description
session.fetch.done		This event indicates the success of the request and that the data location has been updated with the returned content in JSON format.
	requestid	This is the ID of the `<fetch>` request.
	content	This is the returned content. Its format is based on the `<fetch>` request's type attribute. If it is "JSON", the content will be a JSON-based string and the application must use the appropriate function to convert it to the appropriate

Event	Attributes	Description
		ECMAScript objects. The format of the response content will be based on the `<fetch>` type attribute. If there is not a match, an error.session.fetch will be raised. **Note:** when the `<fetch>` method attribute value is "esp", the content value will always be JSON.
	hints	This is the protocol-specific data associated with the fetch response (for example, HTTP header data). Its format is based on the `<fetch>` request's srcexpr and type attributes. If it is HTTP, the content will be ECMAScript Object with the HTTP header elements as properties of the object.
	headers (8.1.200.50)	This is a collection of response headers, presented as key-value pairs. HTTP header data found in the fetch response can be accessed here.
error.session.fetch		This indicates that an error occurred while trying to perform the fetch request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following is a specific error code: • protocol.errorcode - This represents the protocol-specific errors that occur when the attempting the `<fetch>` request.
	description	This is a more detailed description of the error
session.start.done		This event reflects the results of `<start>` and is sent as a confirmation that the session has been started. The sessionid returned shall be the same as the sessionid provided as a return parameter for the sessionid attribute within `<start>`
	sessionid	This is the ID of the SCXML session that has been started.
error.session.start		This indicates that an error occurred while trying to perform the `<start>` request.

Event	Attributes	Description
		The sessionID returned on the action will be invalid after receiving this event.
	sessionid	This is the ID of the SCXML session that was supposed to have started.
	error	This is the type of error that occurred.
	description	This is a more detailed description of the error
session.terminate.done		This event reflects the results of `<terminate>`.
	sessionid	This is the ID of the SCXML session that has terminated.
error.session.terminate		This indicates that an error occurred while trying to perform the `<terminate>` request.
	error	This is the type of error that occurred.
	sessionid	This is the ID associated with a fetch request.
	description	This is a more detailed description of the error
session.updatestart.done		This event reflects the results of `<updatestart>`.
	sessionid	This is the ID of the SCXML session that was updated.
error.session.updatestart		This indicates that an error occurred while trying to perform the `<updatestart>` request.
	error	This is the type of error that occurred.
	description	This is a more detailed description of the error.

Event	Attributes	Description
session.cancel.done		This event reflects the results of `<cancel>`.
	requestid	This is the ID associated with a fetch request.
error.session.cancel		This indicates that an error occurred while trying to perform the `<cancel>` request.
	requestid	This is the ID associated with a fetch request.
	error	This is the type of error that occurred.
	description	This is a more detailed description of the error

The following are the session asynchronous events:

Event	Attributes	Description
done.xxx		This event indicates that a started session was finished or was terminated. The xxx part of the event is different depending on how the session was started. • `<invoke>` - The xxx is "invoke.invokeid" • `<start>` and web service interface initiation - The xxx is "scxml.sessionid" • `<final>` for a state - The xxx is "state.stateid" where statid is the value from the `<state>` id attribute and id is an identifier generated by the platform.
session.restored		This event indicated that a session was restored from a previous checkpoint and some state processing may be lost.
	sessionid	This is the session ID of the session that is being restored.

Event	Attributes	Description
	type	This is the type of event. The only possible value is "external"
session.terminating		This event indicates that the session is being terminated because of a hung condition.This is only sent when a session is being terminated by the platform due to an error condition (hung condition, infinite loop, and so on). This gives the session the ability to graceful terminate itself. So this event is sent by the platform to a session in trouble. A done.xxx will not be sent at all in this condition.
	sessionid	This is the session ID of the session that is being terminated.
	reason	This is the reason the platform is terminating the session. The following is the set of reasons: • TerminationTimeout - The termination cancel operation has not finished in a reasonable time frame or the `<final>` processing for an application has not finished in a reasonable time frame. • IdleTimeout - A session has been idle for a given time period (no events or processing). • ElementCountExceeded - A session has executed too many of the same type of SCXML element (`<transition>`). • ElementTimeout - A session spends too long executing an element (`<script>`, `<queue:submit>`, and so on).
session.cancelled		This event indicates that the session is being cancelled from a `<terminate>` action.
	sessionid	This is the session ID of the session that is being terminated.

In addition to the above listed asynchronous session events, the following events are also available:

- `session.recovered`
- `session.ixnrecovered`
- `session.restarted`

Though the above three events are separate events, they are grouped together as they all signal applications about session run failures on the original ORS instance. They indicate that the session (and as a result the interaction/event processing) was aborted unexpectedly. The reason could be a crash, termination, switchover, or any other factor). And after that processing was resumed on another ORS instance – it could either be the same ORS instance after a restart, or a backup instance switching to the primary mode.

Important

Transitions/handlers for these events (if needed) should be placed in the outermost SCXML state due to the asynchronous nature of these events.

If one of these events is received in a session, the SCXML application could analyze current conditions, and make some corrections in the application logic. The first two events - `session.recovered` and `session.ixnrecovered` – can be seen only if session persistence is enabled and working.The third event - `session.restarted` – does not require session persistence. Moreover, when a session restart is requested in ORS, persistence for this session will be suppressed even if configured.

Event	Attributes	Description
session.recovered		The `session.recovered` event could be received as a result of proactive session recovery upon changing ORS work mode to primary. In order to perform this, in addition to enabling persistence, the SCXML application should be marked as subjected to proactive recovery. In many cases, proactive session recovery is not required. **Important** Proactive session recovery is not recommended for sessions processing multi-media interactions (because of specifics of working with Interaction Server). The `session.recovered` event has no payload/properties, and is sent out-of-band to a proactively recovered session to guarantee it is the very first event in such a session after recovery.

Event	Attributes	Description
session.ixnrecovered		The `session.ixnrecovered` event could be received if a session recovers on a new ORS instance due to an interaction related event. Or, if a session has already been recovered proactively (usually not a case), this event indicates that the new ORS instance restores association between the interaction and the session receiving it. Session recovery by an interaction related event (or recovery on demand) is more frequently used in a live production environment than proactive recovery. In case of session recovery by an interaction related event, it is guaranteed that it is the very first event after restoring the particular session.
	interactionid (of type String)	This is the ID of the interaction re-associated with a given session. Upon processing this event, the restored SCXML session could analyze related interaction properties (for example, user data) and jump to the appropriate SCXML state, or perform other execution logic corrections.
session.restarted		The `session.restarted` event could be received when the Recovery of Voice Calls Without Persistence functionality is configured for a particular or all SCXML applications. It is applicable only for sessions processing voice calls (sessions started by voice call related events). No session persistence is needed at all in this case. When a session restarts, it is guaranteed that the `session.restarted` event is the very first event in the restarted session. The event has no payload/properties. Upon ORS switchover, a new primary ORS instance restarts existing voice sessions if it is requested to do so. For those restarted sessions, the generated events are the same as when a call arrives on the loaded DN – the same set of session startup events (`interaction.added`, `interaction.present`, `interaction.partystatechanged`) are regenerated. There is no event prehistory – session life starts from scratch with one important exclusion. It is guaranteed that the very first event in the restarted session will be the `session.restarted` event. Whereas there is no such event in the original session. A restarted session is expected to raise an internal flag upon processing this event in the same state where the `interaction.added` event is handled. Further, if a flag is set, the session could analyze data when the startup

Event	Attributes	Description
		`interaction.added` event is received and processed, and make a reasonable move to a desired SCXML state to avoid repeating what was already done with the call before restarting the session. Probably, the easiest way to achieve that is setting/checking a milestone mark in call user data. Important User data is not updated upon every SCXML state exit – the milestone mark should be updated upon completion of certain logical units in strategy only. And consequent jumps could be made to re-entrant/independent parts of the strategy. There could also be other ways of making a decision on which part of the strategy to jump to, in a restarted session, if required.

Web Services Interface

Action Elements

<response>

This action is used to send a response to a request-based event from an external application (for details see Send Request to SCXML Session). It is recommended that you use this action element within the `<transition>` element associated with event processing for the given request. If not, you may encounter network-related timeouts and potential performance issues.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	true	value expression	none		This value expression returns the corresponding request ID which this response is for. **Note:** this must be the sendid property from the

Name	Required	Type	Default Value	Valid Values	Description
					associated request event (that is, _event.sendid). See SCXML Legal Data Values and Value Expressions for details.
type	false	value expression	positive	positive negative	This value expression returns the type of response this is. Values defined are: • positive - This indicates that the response is positive. • negative - This indicates that the response is negative. See SCXML Legal Data Values and Value Expressions for details.
resultcode	false	value expression	none	any expression that results in a valid string	This value expression returns a string which will represent the result code associated with the response. See SCXML Legal Data Values and Value Expressions for details.
headers (since 8.1.200.50)	false	value expression	none	any expression that evaluates to an iterable collection of key-value pairs	This value expression defines custom headers (if any) to be included with the HTTP response.

The following is an example of the response processing in the `<transition>` element:

```
<state id="processing_requests_in_transition">
        <transition event="DoFunctionX" cond="_event.data.paramtype == 'application/json' &&
_event.data.param !=''">
```

```
                <script>
                        <! - do specific function x logic ->
                </script>
                <ws:response requestid="_event.sendid">
                        <param name="op1" expr="ovar1"/>
                        <param name="op2" expr="ovar2"/>
                </ws:response>
        </transition>
        <transition event="DoFunctionX" cond="_event.data.paramtype != 'application/json' ||
_event.data.param ==''">
                <ws:response requestid="_event.sendid" type="negative"
resultcode="invalidparameter"/>
        </transition>
</state>
```

The following is an example of the response processing in a sub-state model:

```
<state id="processing_requests_in_substate_model">
        <datamodel>
                <data id="reqid"/>
                <data id="functionXparms"/>
        </datamodel>
        <transition event="DoFunctionX" cond="_event.data.paramtype == 'application/json' &&
_event.data.param !=''" target="functionX'>
                <script>
                        _data.reqid = _event.sendid;
                        _data.functionXparms = _event.data.param;
                </script>
        </transition>
        <transition event="DoFunctionX" cond="_event.data.paramtype != 'application/json' ||
_event.data.param ==''">
                <ws:response requestid="_event.sendid" type="negative"
resultcode="invalidparameter"/>
        </transition>
        <! - This is the substate model to execute the processing associated with function X
->
        <state id="functionX" initial="fXStep1">
                <state id="fXStep1">
                </state>
                ...
                <final>
                        <onentry>
                                <ws:response requestid="_event.sendid">
                                        <param name="op1" expr="ovar1"/>
                                        <param name="op2" expr="ovar2"/>
                                </ws:response>
                        </onentry>
                </final>
        </state>
</state>
```

Children

- `<param>` (Since 8.1.200.25) Occurs 0 to N - This contains data to be passed in the HTTP response.

Events

None

HTTP Mappings

Here is an example of a positive `<response>` action:

```
<ws:response requestid="_event.sendid">
        <param name="param1" expr="'value1'"/>
        <parm name="p2" expr="'v2'"/>
        <parm name="p3" expr="v3"/>
</ws:response>
```

Here is how it maps to an HTTP GET Response message:

```
HTTP/1.1 200
Content-Type=application/json
Content-Length=xx
{"param1":"value1","p2":"v2","p3":{"a":4,"b":5}}
...
```

Here is an example of a negative `<response>` action:

```
<ws:response requestid="_event.sendid" type="negative"
                resultcode="invalidparameter">
        <param name="description" expr="'Invalid value for parm2'"/>
</ws:response>
```

Here is how it maps to an HTTP GET Response message:

```
HTTP/1.1 500 invalidparameter
Content-Type=application/json
Content-Length=xx
{"description":"Invalid value for parm2"}
...
```

Mapping Summary:

- The result of evaluating the `<param>` elements yields the body element formatted in JSON format, where each `<parameter>` name should appear as a top-level attribute. **Note:** the `Content-Length` header value will be set to the total length of the resulting body element.
- The `Content-Type` header value will always be "`application/json`".
- The `Status-Code` header value will either be "`200`" for positive responses or "`500`" for negative responses.
- The `resultcode` attribute will be mapped to the `Reason-Phrase` header element.

Here is an example of a positive `<response>` action:

```
<ws:response requestid="_event.sendid">
        <param name="param1" expr="'value1'"/>
        <parm name="p2" expr="'v2'"/>
        <parm name="p3" expr="v3"/>
</ws:response>
```

Here is how it maps to an HTTP POST Response message:

```
HTTP/1.1 200
Content-Type=application/json
Content-Length=xx
```

```
{"param1":"value1","p2":"v2","p3":{"a":4,"b":5}}
...
```

Here is an example of a negative `<response>` action:

```
<ws:response requestid="_event.sendid" type="negative"
                resultcode="invalidparameter">
        <param name="description" expr="'Invalid value for parm2'"/>
</ws:response>
```

Here is how it maps to an HTTP POST Response message:

```
HTTP/1.1 500 invalidparameter
Content-Type=application/json
Content-Length=xx
{"description":"Invalid value for parm2"}
...
```

Mapping Summary:

- The result of evaluating the `<param>` elements yields the body element formatted in JSON format, where each `<parameter>` name should appear as a top-level attribute. **Note:** the `Content-Length` header value will be set to the total length of the resulting body element.
- The `Content-Type` header value will always be "`application/json`".
- The `Status-Code` header value will either be "`200`" for positive responses or "`500`" for negative responses.
- The `resultcode` attribute will be mapped to the `Reason-Phrase` header element.

Interaction Interface

Behavior Model

Once a session ID is associated with a given interaction, ORS will attach that session ID to the interaction's properties with an "OP-Session-ID" key. This way, if the interaction is redirected or transferred to another resource, the resource will know which orchestration session is associated with this interaction so that it can communicate with it. Note that at any given moment of time, any interaction can be associated with no more than one session. Conversely, several different interactions may be associated with a single session. A caveat arising from this many-to-one relationship is that the deletion of an interaction does not necessarily imply the termination of the associated session.

Associating Interactions

Moving an Interaction's Association from One Session to Another

Typically, an interaction may only be associated with one session at a time. There are certain use cases where an interaction that is currently associated with a session needs to be moved to another session. This process is iniated by the session that is currently associated with the interaction. The following is an example of the process for moving an interaction's association to a different session: When the owning session determines that an interaction that it is working with needs to be associated with another session, the session will associate this interaction using the `<associate>` action. Orchestration Server will do the necessary processing and associate the interaction with the target session by:

1. Adding the interaction to the list of interactions associated with this session (that is, change the _genesys.ixn.interactions[] object)
2. Send the session the appropriate events (interaction.added, interaction.present)
3. Remove the interaction from the original session and send the appropriate events (interaction.deleted, interaction.notcontrolled).

Addressing Resources

The following table indicates how to use the various types of interaction resources when using interaction actions and objects. The resource attribute for these actions and objects can be either a Resource Object or a string:

Resources	string	resource object	Examples:
agents - multi-media)	agent id	resource.agent resource.type = A	string - "agent1" resource object - _data.res.agent
places - (multi-media)	place id	resource.place resource.type = AP	string - "place2" resource object - _data.res.place
agents and group - (voice)	Not supported	resource.agent resource.type = A or GA	string - "agent1" resource object - _data.res.agent
places and place groups - (voice)	Not supported	resource.place resource.type=AP or GP	string - "place2" resource object - _data.res.place
DNs (voice, sip chat)	DN number	resource.dn and resource.switch (optional) resource.type="Q,RP,DN" (optional)	string - "9192340978" resource object - _data.res.dn and _data.res.switch (optional) _data.res.type (optional)
interaction queues (multi-media)	Not supported	resource.type=IQ resource.id = the queue name	resource object - _data.res.id and - data.res.type
workbins	Not supported	resource.type=WB resource.id= the workbin name resource.wb_type= "A, AP, GA, GP" (optional) resource.wb_owner= the name of the workbin owner.	resource object - _data.res.id and _data.res.type and optionally _data.res.wb_type and _data.res.wb_owner.
e-mail addresses	`<username>@<host>` "origin.all" "_origin" indicates that the corresponding address from the related interaction message should be used. "_origin.all" indicates that all the corresponding address from the related interaction message should be used. "_udata" indicates that the corresponding address from the related interaction udata should be used.	Not supported	fred@gmail.com "_origin" "_origin.all" "_udata" "_customerview"
web users	URL	Not supported	http://john.johnson@abc.com
customer numbers	dn number	resource.dn	string - "9192340978" resource object - _data.res.dn
target format addresses	target DN	Resource object from the queue.submit.done event	12345@switch3.DN

Object Model

See Interaction Interface Object Model

Functions

See Interaction Interface Functions

Action Elements

See Interaction Interface Action Elements

Events

See Interaction Interface Events

Interaction Interface Object Model

The following are the ECMAScript objects for the interaction model interface. **Notes**:

- The object model is changed only in results of events. Every interaction event can result in a change in content of the "object model".
- After the session is started, it does not have any interactions - the _genesys.ixn.interactions[] array is empty (even if the platform started the session at the request of an interaction).
- Before accessing or manipulating interactions, the application logic must be sure that session has at least one interaction in its data (that is, the _genesys.ixn.interactions[] array is not empty). For example, the session needs to wait for an event like interaction.added before trying to use the interaction array.
- Attempts to access interactions in the object model before they are available will result in a runtime exception in the session, with the corresponding error event.

_genesys.ixn Object

Each SCXML session will have an association with the Interaction functional module (if an interaction is involved with the session). This object allows the session to access the set of interaction-related objects and properties that are associated with the given SCXML session. The name of the object will be "_genesys.ixn". This object is accessible through the _genesys.FMname property.

Name	Access	Type	Default Value	Valid Values	Description
interactions	read only	array of interaction objects	none		This is the list of interactions currently associated with the logic. Each interaction will be represented by an ECMAScript object owned by the Interaction functional module. For routing strategy-based logic only the first entry in the list (interactions[0]) will be used. This list is maintained by the orchestration platform based on the interaction between the orchestration logic and the interaction-related functional modules. So when an interaction is associated with the orchestration logic through an event or action, it is added to the list and when the interaction ends it is removed from the list. This property (list only) is read only. See section interaction object for details on the interaction object.

_genesys.ixn.mediaType ENUM Object

This represents the media type enumeration.

Name	Access	Type	Default Value	Valid Values	Description
TMediaVoice	read only	integer	none	0	The media for the interaction is voice.
TMediaVoIP	read only	integer	none	1	The media for the interaction is VoIP.
TMediaEMail	read only	integer	none	2	The media for the interaction is e-mail.
TMediaVMail	read only	integer	none	3	The media for the interaction is voice mail.
TMediaSMail	read only	integer	none	4	The media for the interaction is snail mail.
TMediaChat	read only	integer	none	5	The media for the interaction is chat.
TMediaVideo	read only	integer	none	6	The media for the interaction is video.
TMediaCobrowsing	read only	integer	none	7	The media for the interaction is co -browse.
TMediaWhiteboard	read only	integer	none	8	The media for the interaction is whiteboard.
TMediaAppSharing	read only	integer	none	9	The media for the interaction is application sharing.
TMediaWebform	read only	integer	none	10	The media for the interaction is web form.
TMediaWorkItem	read only	integer	none	11	The media for the

Name	Access	Type	Default Value	Valid Values	Description
					interaction is work item.
TMediaCallback	read only	integer	none	12	The media for the interaction is callback.
TMediaFax	read only	integer	none	13	The media for the interaction is fax.
TMediaIMChat	read only	integer	none	14	The media for the interaction is IM.
TMediaBusinessEvent	read only	integer	none	15	The media for the interaction is business event.
TMediaAlert	read only	integer	none	16	The media for the interaction is alert.
TMediaSMS	read only	integer	none	17	The media for the interaction is SMS.
TMediaOutboundPreview	read only	integer	none	18	The media for the interaction is outbound preview.
TMediaOpenMedia	read only	integer	none	19	The media for the interaction is open media item.
TMediaNativeSMS	read only	integer	none	20	The media for the interaction is native SMS.

_genesys.ixn.callState ENUM Object (since 8.1.3)

This represents the call state type enumeration.

Name	Access	Type	Default Value	Valid Values
Ok	read only	integer	none	0

Name	Access	Type	Default Value	Valid Values
Transferred	read only	integer	none	1
Conferenced	read only	integer	none	2
GeneralError	read only	integer	none	3
SystemError	read only	integer	none	4
RemoteRelease	read only	integer	none	5
Busy	read only	integer	none	6
NoAnswer	read only	integer	none	7
SitDetected	read only	integer	none	8
AnsweringMachineDetected	read only	integer	none	9
AllTrunksBusy	read only	integer	none	10
SitInvalidnum	read only	integer	none	11
SitVacant	read only	integer	none	12
SitIntercept	read only	integer	none	13
SitUnknown	read only	integer	none	14
SitNocircuit	read only	integer	none	15
SitReorder	read only	integer	none	16
FaxDetected	read only	integer	none	17
QueueFull	read only	integer	none	18
Cleared	read only	integer	none	19

Name	Access	Type	Default Value	Valid Values
Overflowed	read only	integer	none	20
Abandoned	read only	integer	none	21
Redirected	read only	integer	none	22
Forwarded	read only	integer	none	23
Consult	read only	integer	none	24
Pickedup	read only	integer	none	25
Dropped	read only	integer	none	26
Droppednoanswer	read only	integer	none	27
Unknown	read only	integer	none	28
Covered	read only	integer	none	29
ConverseOn	read only	integer	none	30
Bridged	read only	integer	none	31
SilenceDetected	read only	integer	none	32
Deafened	read only	integer	none	49
Held	read only	integer	none	50

Interaction Objects

interaction Object

Each interaction associated with a given SCXML session will have an object to represent the common properties of an interaction. This object and its properties are maintained by the functional module, but certain properties can be set or updated by the orchestration logic itself. The name of the object will be "interaction". This object is accessible through the _genesys.FMname.interactions[] property. This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
g_uid	read only	string	none		This is the globally unique ID for the interaction that is defined by the underlying media system. • T-Server - g_uid • Interaction Server - attr_itx_id
category	read only	string	none	voice, msgbased, chat	This is the media category associated with the interaction. It defines the type of media extension that is associated with the interaction.
tenantid	read only	string	none		This is the ID of the tenant that this interaction was originated from.
parentid	read only	string	none		This is the globally unique ID of this interaction's parent interaction.
contactedaddr	read only	string	none	Any valid string that represents the address	This is the address of the resource that was initially contacted and

Name	Access	Type	Default Value	Valid Values	Description
					started this interaction. This property will also be represented in a media- or channel-specific property. For example: • `_genesys.ixn.interactio` for voice interactions • `_genesys.ixn.interactio` for msgbased interactions
parties	read only	array of party objects	none		This is the list of parties or resources currently associated with the interaction. Each party will be represented by an ECMAScript object owned by the Interaction functional module.
udata	read only	object	none	Any valid ECMAScript object	This is application data that can be associated with and attached to the interaction so that the coordination of processing between resources is seamless. An application will manage this data through this property and the defined set of functions. For example, • Get udata kvpair value - `xvalue = _genesys.ixn.interactio` • Set or add udata kvpair - `_genesys.ixn.setuData(a` • Remove

Name	Access	Type	Default Value	Valid Values	Description
					udata kvpair - `_genesys.ixn.deleteuDat` As a result of these actions, the appropriate action will be taken on the underlying Interaction functional module system (for example, T-Server and user data). The udata property does not support ECMAScript arrays either as a value of the udata property or as a property of any object in the tree. This will be used not only for user data, but also for URS-based business data and interaction data.
voice	read only	voice object			This is the object that contains the voice extensions to the interaction. NOTE: This property only exists for voice interactions.
msgbased	read only	msgbased object			This is the object that contains the msgbased extensions to the interaction. NOTE: This property only exists for message based interactions.
chat	read only	chat object			This is the object that contains the chat extensions to the interaction. NOTE: This property only exists for chat interactions.
xdata	read only	object	none	Any valid ECMAScript object	This is extension data that has been associated with the interaction as a

Name	Access	Type	Default Value	Valid Values	Description
					result of the event that started this session and is read only. An application may access extension data through this property. For example, Get xdata kvpair value - x `value = _genesys.ixn.interactions[x].xda`
location	read only	object	none		This is the location property of the interaction. This object conists of the following properties: control_server media_server

party Object

Each party or business resource involved in the associated interaction will be represented by an object and a common set of properties. These objects and their properties are maintained by the functional module, but certain properties can be set or updated by the orchestration logic itself. The name of the object will be "parties" and is accessible through the interaction object. This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
g_uid	read only	string	none		This is the globally unique ID for the party that is defined by the underlying media system.
interactionid	read only	string	none		This is the globally unique ID of the interaction this party belongs to.
devicetype	read only	unknown	**For voice:** agent, queue, routepoint, treatmentport, unknown **For multimedia:** queue, unknown **For non-voice**		This is the general type of device associated with this interaction.

Name	Access	Type	Default Value	Valid Values	Description
			and non-multimedia: customer		
device	read only	string	null		This is the device associated with this interaction party.
resource	read only	string	null		This is the resource ID associated with this interaction party.
voicep	read only	voicep object			This is the object that contains the voice extensions to the party. NOTE: This property only exists for voice interactions.
msgbasedp	read only	msgbasedp object			This is the object that contains the msgbased extensions to the party. NOTE: This property only exists for message based interactions.
chatp	read only	chatp object			This is the object that contains the chat extensions to the party. NOTE: This property only exists for chat interactions.

Voice Objects

voice Object

The following is the ECMAScript object which contains the interaction extensions for voice-related interactions. Properties of that object may be updated during interaction lifetime, for example, when interaction parties have been added/removed/changed, so keep that in mind when you decide to use them. This object is accessible through the _genesys.FMname.interactions[].voice property. These properties can be dynamically accessed using the following format: _genesys.FMname.interactions[x].[_genesys.FMname.interactions[x].category].xxx. They are also accessible via actions or events. The following are the voice extension properties of the interaction object.

Name	Access	Type	Default Value	Valid Values	Description
type	read only	string	none	unknown, internal, inbound, outbound, consult, callback	This is the origin type of the interaction.
media	read only	string	none	The following properties from _genesys.ixn.mediaType object: TMediaAny, TMediaCallback, TMediaOutboundPreview, TMediaVideo, TMediaVMail, TMediaVoice, TMediaVoIP	This is the originating media type of the interaction.
ani	read only	string	none		This is the ANI associated with the calling party.
dnis	read only	string	none		This is the DNIS associated with phone number that the customer called.
ced	read only	string	none		This is the last set of digits collected from the caller. NOTE: This property is populated from the event data, which is optional. If digits were collected, then this

Name	Access	Type	Default Value	Valid Values	Description
					propertly is set.
acdq	read only	string	none		This is the ACD queue that this interaction is or was queued in. NOTE: This property is populated from the event data, which is optional. If ACD queue was used, then this propertly is set.
callid	read only	string	none		This is the callid created by the switch.
connid	read only	string	none		This is the connection ID generated for this interaction by the underlying media system (that is, T-Server).

voicep Object

The following are the voice-specific values for the party object.

Name	Access	Type	Default Value	Valid Values	Description
state	read only	string	none	null initiated alerting connected hold queued fail	This is the state of the party in relationship to the interaction.

Message Objects

msgbased Object

The following is the ECMAScript object which contains the interaction extensions for msgbased-related interactions. This object is accessible through the _genesys.FMname.interactions[].msgbased property. These properties can be dynamically accessed using the following format: _genesys.FMname.interactions[x].[_genesys.FMname.interactions[x].category].xxx. They are also accessible via actions or events. The following are the msgbased extension properties of the interaction object.

Name	Access	Type	Default Value	Valid Values	Description
type	read only	string	none	The list of valid values are those that are defined in configuration server under: Business Attributes / Interaction Attributes / Attributes Values	This is the origin type of the interaction.
state	read only	string	none	queued, cached, routing, handling, unknown	This is the state of the interaction.
media	read only	string	none	The following properties from _genesys.ixn.mediaType object: TMediaAny, TMediaEMail, TMediaFax, TMediaSMail, TMediaNativeSMS, TMediaSMS, TMediaWebForm, TMediaOpenMedia	This is the originating media type of the interaction.
from	read only	URI	none	Any valid string or Resource Object	This is the address that the message came from.
to	read only	array of URIs	none	Any valid string or Resource Object	This is the list of addresses that this message was sent to
cc	read only	array of URIs	none	Any valid string or	This is the list of

Name	Access	Type	Default Value	Valid Values	Description
				Resource Object	addresses that were copied on this message.
queue	read only	string	none	Any valid string	This is the current queue name associated with this interaction.
view	read only	string	none	Any valid string	This is the current view associated with this interaction. **First appears in Orchestration release 8.1.2**
subject	read only	string	none	Any valid string	This is the subject line of the associated message.
content	read only	content object	none	none	• • • This property is not yet supported*** This is the content of the message itself.
received_at	read only	string	none	none	Value of "_attr_itx_received_at" interaction attribute
submitted_at	read only	string	none	none	Value of "_attr_itx_submitted_at" interaction attribute
placed_in_queue_at	read only	string	none	none	Value of "_attr_itx_placed_in_queue_at" interaction attribute
is_online	read only	boolean	none	none	Value of "_attr_itx_is_online" interaction attribute

Name	Access	Type	Default Value	Valid Values	Description
is_locked	read only	boolean	none	none	Value of "_attr_itx_is_locked" interaction attribute
moved_to_queue_at	read only	string	none	none	Value of "_attr_itx_moved_to_queue_at" interaction attribute
externalID	read only	string	none	none	This is the ID of the interaction that has been assigned by the originating media server.

msgbasedp Object

The following are the msgbased-specific values for following the party object.

Name	Access	Type	Default Value	Valid Values	Description
state	read only	string	none	null initiated alerting connected hold queued fail	This is the state of the party in relationship to the interaction.

content Object

The following are the msgbased-specific values for following the message content object.

Name	Access	Type	Default Value	Valid Values	Description
btype	read only	string	none	application/ msword, application/octet-stream, application/ postscript, application/rtf, application/ vnd.ms-powerpoint, application/	This is a string that specifies the MIME type of the binary content.

Name	Access	Type	Default Value	Valid Values	Description
				vnd.ms-project, application/vnd.visio, application/voicexml+xml, application/xml, application/xml-dtd, application/zip, audio/basic, audio/mpeg, audio/mpeg4- generic, image/g3-fax, image/gif, image/jpeg, image/tiff, message/delivery-status, message/http, message/news, message/partial, message/rfc822, message/sip, message/sipfrag, message/tracking-status, multipart/alternative, multipart/form-data, multipart/mixed, multipart/parallel, multipart/voice-message, text/html, text/plain, text/richtext, text/xml, video/DV, video/JPEG, video/MPEG, video/mpeg4-generic, video/quicktime, video/raw	
binary	ready only	binary	none	none	This is the complete content of the message (for example, raw text plus MIME content, if any, plus attached files if any).
structuredtext	ready only	string	none	none	This is only the structured content of the message.
text	read only	string	none		This is only the raw text content (unformatted and unstructured) of the message

Chat Objects

chat Object

The following is the ECMAScript object which contains the interaction extensions for chat-related interactions. This object is accessible through the _genesys.ixn.interactions[].chat property. These properties can be dynamically accessed using the following format: _genesys.ixn.interactions[x].[_genesys.FMname.interactions[x].category].xxx. They are also accessible via actions or events. The following are the chat extension properties for the interaction object.

Name	Access	Type	Default Value	Valid Values	Description
queue	read only	string	none	any valid string	This is the current queue name associated with this interaction.
type	read only	string	none	unknown, chat, chatrequest, cobrowse,	This is the origin type of the interaction.
state	read only	string	none	queued, cached, routing, handling, unknown	This is the state of the interaction.
media	read only	string	none	The following properties from the _genesys.ixn.mediaType object: TMediaAny, TMediaChat, TMediaCoBrowsing, TMediaIMChat	This is the originating media type of the interaction.
received_at	read only	string	none	none	Value of "_attr_itx_received_at" interaction attribute
submitted_at	read only	string	none	none	Value of "_attr_itx_submitted_at" interaction attribute
placed_in_queue_at	read only	string	none	none	Value of "_attr_itx_placed_in_queue_at" interaction attribute

Name	Access	Type	Default Value	Valid Values	Description
is_online	read only	boolean	none	none	Value of "_attr_itx_is_online" interaction attribute
is_locked	read only	boolean	none	none	Value of "_attr_itx_is_locked" interaction attribute
moved_to_queue_at	read only	string	none	none	Value of "_attr_itx_moved_to_queue_at" interaction attribute
externalID	read only	string	none	none	This is the ID of the interaction that has been assigned by the originating media server.

chatp Object

The following are the chat-specific values for the following party object.

Name	Access	Type	Default Value	Valid Values	Description
state	read only	string	none	null initiated alerting connected hold queued fail	This is the state of the party in relationship to the interaction.

chatmessage Object

The following are the chat message object properties:

Name	Access	Type	Default Value	Valid Values	Description
date	read only	integer	none	none	This is the number of seconds since 1/1/1970.

Name	Access	Type	Default Value	Valid Values	Description
device	read only	URI	none	Any valid string or Resource Object.	This is the device address of the party that created the message.
text	read only	string	none	none	This is the text message that was sent.

Interaction Interface Functions

```
= Functions =
```

_genesys.ixn.getMediaIntValue

This function returns the integer value for the interaction media type name based on the string name. `intvalue _genesys.ixn.getMediaIntValue(media)` Parameters:

- **media:** STRING which can be a variable or a constant - This parameter is the string form of the media type name.

Returns: `intvalue`: NUMBER - The result of the function is an integer value which represents the given media type. A value of -1 will indicate that an integer value could not be found (before 8.1.200.50, the default value was 0 when the value was not found). The following is an example of the function:

```
  <state id="check">
    <onentry>
        <log expr="'This function returns the integer value for the interaction media type'"
/>
        <script>
                if ( _genesys.ixn.getMediaIntValue( 'TMediaVoice' ) == 0 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaVoIP' ) == 1 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaEMail' ) == 2 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaVMail' ) == 3 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaSMail' ) == 4 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaChat' ) == 5 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaVideo' ) == 6 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaCobrowsing' ) == 7 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaWhiteboard' ) == 8 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaAppSharing' ) == 9 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaWebform' ) == 10 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaWorkItem' ) == 11 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaCallback' ) == 12 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaFax' ) == 13 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaIMChat' ) == 14 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaBusinessEvent' ) == 15 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaAlert' ) == 16 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaSMS' ) == 17 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaOutboundPreview' ) == 18 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaOpenMedia' ) == 19 &&
                         _genesys.ixn.getMediaIntValue( 'TMediaNativeSMS' ) == 20 &&
                         _genesys.ixn.getMediaIntValue( 'unknown' ) == -1 &&
                         typeof( _genesys.ixn.getMediaIntValue( 'TMediaVoice' ) ) == 'number')
                {
                        _data.testPassed = 1;
                }
        </script>
    </onentry>
    <transition cond="_data.testPassed==1" target="routing"/>
    <transition cond="_data.testPassed==0" target="error" >
                <log expr="uneval( _event )" />
    </transition>
  </state>
```

_genesys.ixn.setuData

This function adds new udata or updates existing udata for an interaction. This function will not affect existing udata values that do not match the property names defined in the input parameter. For example:

Existing udata	Requested additions and updates	Resulting udata
udata.d1 = 3 udata.d2 = 4 udata.d3 = 5	d3 = 10000 d6 = 67 d7.a = 88 d7.b =99	udata.d1 = 3 udata.d2 = 4 udata.d3 = 10000 udata.d6 = 67 udata.d7.a = 88 udata.d7.b = 99

`void _genesys.ixn.setuData(input, ixnid)` Parameters:

- **input:** OBJECT which must be a variable - This parameter is an object which CONTAINs the properties which are to be added to the udata of the given interaction. This means that the input object itself is NOT added to the interaction's udata (for example, if the input parameter object is "a", with properties "b=7" and "c=9" then the resulting udata properties for the interaction will be `_genesys.ixn.interactions[_data.ixnid].udata.b` and `_genesys.ixn.interactions[_data.ixnid].udata.c`). The following is an example `_genesys.ixn.setuData({b:7, c:9});`
- **ixnid:** STRING which can be a variable - This parameter is optional. It defines the ID of the interaction which should have its udata added to or updated.

Returns: void The following is an example of this function:

```
  <state id="setudata">
    <onentry>
      <script>
        var data = { details : { name : "Smith, John", age : 45 } };
        _genesys.ixn.setuData( data );
        _genesys.ixn.setuData( { category : 1 } );
      </script>
    </onentry>
    <transition target="update"/>
  </state>
  <state id="update">
    <onentry>
      <script>
        _genesys.ixn.setuData( { category : 2 }, _data.ixnid );
      </script>
    </onentry>
    <transition target="check"/>
  </state>

  <state id="check">
    <transition event="interaction.udata.changed"
cond="_genesys.ixn.interactions[_data.ixnid].udata.category==2 &&

_genesys.ixn.interactions[_data.ixnid].udata.details.name=='Smith, John' &&
_genesys.ixn.interactions[_data.ixnid].udata.details.age==45" target="routing" />
  </state>
```

_genesys.ixn.deleteuData

This function deletes a udata property or all of the udata properties from the given interaction. `void _genesys.ixn.deleteuData(key, ixnid)` Parameters:

- **key:** STRING (variable or constant) or JavaScript object (variable or object literal) - This parameter may be a STRING that represents the key or object, which represents a subset of udata properties that are to be deleted from udata (see example below).
- **ixnid:** STRING, which can be a variable - This parameter is optional. It defines the ID of the interaction which should have a given udata property removed.

Returns: `void` The following is an example of the function:

```
<state id="setudata">
    <onentry>
        <script>
                var data = { details : { name : "Smith, John", age : 45 } };
                _genesys.ixn.setuData( data );
      </script>
    </onentry>
    <transition event="interaction.udata.changed"
                cond="_genesys.ixn.interactions[_data.ixnid].udata.details.name=='Smith,
John' &&
                       _genesys.ixn.interactions[_data.ixnid].udata.details.age==45"
                target="delete"/>
</state>
<state id="delete">
    <onentry>
        <script>
                _genesys.ixn.deleteuData( { details:{ age:0 } }, _data.ixnid );
        </script>
    </onentry>
    <transition event="interaction.udata.changed"
                cond="_genesys.ixn.interactions[_data.ixnid].udata.details.name=='Smith,
John' &&
                        _genesys.ixn.interactions[_data.ixnid].udata.details.age==undefined"
                target="delete_all_udata"/>
</state>

<state id="delete_all_udata">
    <onentry>
        <script>
                _genesys.ixn.deleteuData( { $ALL: 0 } );
        </script>
    </onentry>
    <transition event="interaction.udata.changed"
                cond="_genesys.ixn.interactions[_data.ixnid].udata.ORSession==undefined &&
                        _genesys.ixn.interactions[_data.ixnid].udata.ORDbid==undefined &&
                        _genesys.ixn.interactions[_data.ixnid].udata.ORUrl==undefined &&
                        _genesys.ixn.interactions[_data.ixnid].udata.details==undefined"
                target="routing"/>
</state>
```

Interaction Interface Action Elements

Common Actions

The following are the common actions across interactions:

<terminate>

This is the action to terminate the interaction. It is equivalent to the `TClearCall` request in T-Server. For non-voice interactions, this action can only be done when the interaction is "presented" with *RequestStopProcessing*.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique

Name	Required	Type	Default Value	Valid Values	Description
					identifier.
interactionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
reason	false	value expression	none	Any value expression that returns a valid string	A value expression which returns a character string which identifies the reason why the interaction is being terminated. See SCXML Legal Data Values and Value Expressions for details.
resource	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the resource address for the call to be terminated. Has no meaning when used with a multimedia interaction. See SCXML Legal Data Values and Value Expressions **Note**: Since 8.1.100.19
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when redirecting this interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal

Name	Required	Type	Default Value	Valid Values	Description
					Data Values and Value Expressions for details.

hints Attribute Considerations

- When the interaction is non-voice, properties "ucs" and "delete" could be defined in "hints" object.
    - "ucs" - if true, OCS will also send ESP request with Method=StopProcessing via Ixn Server to UCS.
    - "delete" - value of this property will be attached as "Delete" parameter to ESP request.

The following is an example:

```
<state id="do_terminate">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:terminate requestid="_data.reqid" interactionid="_data.ixnid"
             reason="'finished service X'"
resource="_genesys.ixn.interactions[_data.ixnid].voice.dnis"/>
</onentry>
<transition event="interaction.terminate.done" target="statex"/>
<transition event="error.interaction.terminate" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `interaction.terminate.done` - This event is sent when the request has been accepted by the system and the interaction has started the termination process.
- `error.interaction.terminate` - This event is sent when the request itself has failed for some reason.
- `interaction.deleted` - This event is sent when the interaction is actually terminated.

<clear>

This is the action to clear a given party from the interaction. Note that in cases of conference interactions, the interaction will remain alive until there is no more than one party in the interaction. This is equivalent to the `TReleaseCall` request in T-Server. **Note:** This action is only for voice interactions at this time, but will apply to other media in the future.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
resource	true	value expression	none	Any valid string or Resource Object	A value expression which returns the resource address currently involved in the interaction for which the connection will be cleared. See SCXML Legal Data

Name	Required	Type	Default Value	Valid Values	Description
					Values and Value Expressions for details.
reason	false	value expression	none	Any value expression that returns a valid string	A value expression which returns a character string which identifies the reason why the resource connection with the interaction is being cleared. See SCXML Legal Data Values and Value Expressions for details.

The following is an example:

```
<state id="do_clear">
<datamodel>
        <data id="reqid"/>
        <data id="ixnid"/>
        <data id="rscid"/>
</datamodel>
<onentry>
        <ixn:clear requestid="_data.reqid" interactionid="_data.ixnid" resource="_data.rscid"
reason="'finished service X'"/>
</onentry>
<transition event="interaction.clear.done" target="statex"/>
<transition event="error.interaction.clear" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `interaction.clear.done` - This event is sent when the request has been accepted by the system and the resource's connection to the interaction has started the clearing process.
- `error.interaction.clear` - This event is sent when the request itself has failed for some reason.
- `interaction.partydeleted` - This event is sent when the resources's connection to the interaction is actually cleared.

<redirect>

This is the action to redirect the interaction to another resource. This action is used for the following media server actions:

- T-Server - `TRouteCall`, (IRD Functions - DeliverCall, DeliverToIVR, RouteCall, TRoute)
- Interaction Server - `RequestPlaceInQueue, RequestPlaceInWorkbin, RequestDeliver` (IRD Function Blocks - Queue Interaction, IRD Functions - RouteCall, TRoute)

When using the `<redirect>` call, the redirected interaction may or may not start a new session depending on whether or not the redirecting session is already terminated. In the event that the interaction arrives on the target resource after the redirecting session has terminated, a new session will be started. Otherwise, if the interaction arrives on the target resource prior to the termination of the redirecting session, no new session will be created. To guarantee the creation of a new session for the redirected interaction, it is suggested to use `<detach>` to dissolve the interaction/session association.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
detach	false	Boolean expression	false	true, false	ORS Release 8.1.400.21 introduced a detach attribute for the <ixn:redirect> action. The default value is false to comply with existing/previous ORS behavior. The attribute controls whether ORS should detach an interaction from the current session before routing to the specified target, which can allow ORS to start processing the next session. As further detailed below, if the value is true, then the interaction is detached from the session before routing to the specified target. If the value is false, then no detach occurs before routing. **Succesful Redirect/ detach=true** In the case of a successful execution of a redirect action with detach=true, a routed interaction (previously belonging to the

Name	Required	Type	Default Value	Valid Values	Description
					session that invoked this action) will be automatically detached. The SCXML session invoking this action receives interaction.notcontrolled and interaction.deleted events with resultof=detaching, and then interaction.redirect.done event. **Unsuccesesful Redirect/ detach=true** In the case of a failed <ixn:redirect> action (with detach=true), the session invoking this action receives only the error.interaction.redirect event. The interaction remains in the same attached/detached state as it was before invoking <ixn:redirect> action with detach=true. **Note:** If the <ixn:redirect> action is provided with the ID of an interaction that is already detached, then the detach attribute has no effect. In this case, ORS produces warning IxnVMMThreadData::CommonPartHan WARNING - ixn action attribute 'detach' has no effect, interaction 'ZZZ' is already detached and operates as if detach=false. If a redirect error happens, the interaction remains in the same state

Name	Required	Type	Default Value	Valid Values	Description
					(is still detached) and the application developer must decide the next step for the interaction.
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
from	true	value expression	none	Any valid string or Resource Object	A value expression which returns the address that this interaction is to be

Name	Required	Type	Default Value	Valid Values	Description
					redirected from. For details, on the format, see Addressing Resources. See SCXML Legal Data Values and Value Expressions
to	true	value expression	none	Any valid string or Resource Object	A value expression which returns the destination address for this interaction. For details, on the format, see Addressing Resources. See SCXML Legal Data Values and Value Expressions
type	false	NMTOKEN	_genesys.queue.rType.RouteTypeDefault	Values from the _genesys.queue.rType enumeration	This defines the type of redirection processing that is to be done.
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when redirecting this interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details.

hints Attribute Considerations Note: you can use a special interaction queue name (`__STOP__`) which will do the processing required to stop the interaction.

- When the interaction is non-voice, inPersistentQueues and OutPersistentQueues properties of the Interaction Server interaction will be set automatically to the appropriate queues defined for the session (this information comes from the trigger information). So these properties do not need to be exposed. BUT if the developer wants to explicitly specify these properties, they can supply them through the hints attribute object. The object should have the following properties

- "inqueues" - This is an ECMAScript object with a collection of string properties. The name of the property is an interaction queue name and the value is a description of that queue.
- "outqueues" - This is an ECMAScript object with a collection of string properties. The name of the property is an interaction queue name and the value is a description of that queue.

- Property **extensions** of **hints** object allows to specify the content of `AttributeExtension` for the corresponding `TRequestRouteCall` request. A valid value for property **extensions** is an ECMAScript object. (Since 8.1.2)
- Property **reasons** of **hints** object allows to specify the content of `AttributeReason` for the corresponding `TRequestRouteCall` request. A valid value for property **reasons** is an ECMAScript object. (Since 8.1.4)

The following are some examples:

```
<state id="do_redirect_voice_interaction_to_an_agent_dn">
   <onentry>
        <ixn:redirect interactionid="_data.ixnid" from="'1010'" to="'1111'"/>
   </onentry>
   <transition event="interaction.redirect.done" target="statex"/>
   <transition event="error.interaction.redirect" target="statey"/>
</state>
<state id="do_redirect_mm_interaction_to_an_agent">
   <onentry>
        <ixn:redirect interactionid="_data.ixnid" from="'1010'" to="'1111'"/>
   </onentry>
   <transition event="interaction.redirect.done" target="statex"/>
   <transition event="error.interaction.redirect" target="statey"/>
</state>
<state id="do_redirect_mm_interaction_to_an_interaction_queue">
   <onentry>
        <ixn:redirect requestid="_data.reqid" to="'queueF'"/>
   </onentry>
   <transition event="interaction.redirect.done" target="statex"/>
   <transition event="error.interaction.redirect" target="statey"/>
</state>
<state id="do_redirect_multi-resource_interaction_to_another_agent">
   <onentry>
        <ixn:redirect requestid="_data.reqid" interactionid="_data.ixnid"
                      from="'3444'" to="'6555'"/>
   </onentry>
   <transition event="interaction.redirect.done" target="statex"/>
   <transition event="error.interaction.redirect" target="statey"/>
</state>
<state id="do_redirect_voice_interaction_to_another_queue-route_point">
   <onentry>
        <ixn:redirect requestid="_data.reqid" to="'6666'"/>
   </onentry>
   <transition event="interaction.redirect.done" target="statex"/>
   <transition event="error.interaction.redirect" target="statey"/>
</state>
<state id="do_redirect_with_resource_object">
   <onentry>
        <script>
                var dest = {type:"A",
                            agent:"702_sip",
                            dn:"702",
                            id:"702_sip",
                            place:"702",
                            'switch':"SipSwitch"};
                var myhints = { extensions : { ext1:"extval1" }, reasons : {reason1:2} };
        </script>
        <ixn:redirect requestid="_data.reqid" interactionid="_data.ixnid"
                      from="'RP_sip1'" to="dest" type="1" hints="myhints" />
   </onentry>
   <transition event="interaction.redirect.done" target="statex"/>
   <transition event="error.interaction.redirect" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `interaction.redirect.done` - This event is sent when the request has been accepted by the system and the interaction has started the redirection process.
- `error.interaction.redirect` - This event is sent when the request itself has failed for some reason.
- `interaction.ondivert` - This event is sent when the interaction has been moved to the defined destination.

<singlesteptransfer>

This is the action to transfer the interaction to another resource in a single step. This is equivalent to the `TSingleStepTransfer` request in T-Server. **Note:** This action is only for voice interactions at this time

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not

Name	Required	Type	Default Value	Valid Values	Description
					specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
from	true	value expression	none	Any valid string or Resource Object.	A value expression which returns the address that this interaction is to be transferred from. For details, on the format, see Addressing Resources. See SCXML Legal Data Values and Value Expressions
to	true	value expression	none	Any valid string or Resource Object.	A value expression which returns the destination address for this interaction. For details, on the format, see Addressing Resources. See SCXML Legal Data Values and Value Expressions
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object

Name	Required	Type	Default Value	Valid Values	Description
					containing information which may be used by the implementing functional module when transferring this interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details.

The following are some examples:

```
<state id="dosingle_step_transfer_voice_interaction_to_an_agent_dn">
<datamodel>
        <data id="reqid"/>
        <data id="ixnid"/>
        <data id="rscid"/>
</datamodel>
<onentry>
        <script>
                var myhints = { extensions:{ key_1:'Value1', key_2:200*200, key_3:300*300 } };
        </script>
        <ixn:singlesteptransfer requestid="_data.reqid" interactionid="_data.ixnid"
from="_data.rscid" to="'1111'" hints="myhints" />
</onentry>
<transition event="interaction.singlesteptransfer.done" target="statex"/>
<transition event="error.interaction.singlesteptransfer" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `interaction.singlesteptransfer.done` - This event is sent when the request has been accepted by the system and the interaction has started the single step transfer process.
- `error.interaction.singlesteptransfer` - This event is sent when the request itself has failed for some reason.
- `interaction.onsinglesteptransfer` - This event is sent when the interaction has been transferred to the defined destination.

<singlestepconference>

This is the action to conference another resource into the interaction in a single step. It is equivalent to the TSingleStepConference request in T-Server. **Note:** This action is only for voice interactions at this time

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must

Name	Required	Type	Default Value	Valid Values	Description
					receive a unique identifier.
interactionid	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
from	true	value expression	none	Any valid string or Resource Object	A value expression which returns the address that this interaction is to be conferenced from. For details, on the format, see Addressing Resources. See SCXML Legal Data Values and Value Expressions
to	true	value expression	none	Any valid string or Resource Object.	A value expression which returns the destination address for the resource that is to be conferenced into the interaction. For details, on the format, see Addressing Resources. See SCXML Legal Data Values and Value Expressions

The following is an example:

```
<state id="dosingle_step_conference_voice_interaction_to_an_agent_dn">
<datamodel>
        <data id="reqid"/>
        <data id="ixnid"/>
        <data id="rscid"/>
</datamodel>
<onentry>
        <ixn:singlestepconference requestid="_data.reqid" interactionid="_data.ixnid"
from="_data.rscid" to="'1111'"/>
</onentry>
<transition event="interaction.singlestepconference.done" target="statex"/>
<transition event="error.interaction.singlestepconference" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `interaction.singlestepconference.done` - This event is sent when the request has been accepted by the system and the interaction has started the single step conference process.
- `error.interaction.singlestepconference` - This event is sent when the request itself has failed for some reason.
- `interaction.onsinglestepconference` - This event is sent when the specified resource has been conferenced into the interaction.

<associate>

This is the action to associate an interaction with the target SCXML session and un-associate it with the current session. This is implicitly done when a session is triggered by an interaction event, but there are cases where an interaction that is currently not associated with another SCXML session needs to be associated with it by the session that currently owns the interaction. The association process will add the interaction to the target session's `_genesys.ixn.interactions[]` array. In addition, the functional module will do the necessary media-specific processing to ensure that this session will be able to receive events associated with this interaction as well as be able to take action against it. This may also include the taking of ownership of this interaction from another SCXML session or 3rd party application. This level of processing will depend on the capabilities of the underlying functional module and its associated system.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is

Name	Required	Type	Default Value	Valid Values	Description
					dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
category	true	value expression	none	Any value expression that returns one of the following values: voice, msgbased, chat, web	A value expression which returns the category for this interaction. See SCXML Legal Data Values and Value Expressions for details.
Interactionid	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the interaction id that is to be associated with this session. See SCXML Legal Data Values and Value Expressions for details.
sessionid	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the session id to associate interaction with. See SCXML Legal Data Values and Value Expressions for details.

The following is an example:

```
<state id="do_associate">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:associate requestid="_data.reqid" category="'voice'"
interactionid="_event.data.id" sessionid="'12345678'"/>
</onentry>
<transition event="interaction.associate.done" target="statex"/>
<transition event="error.interaction.associate" target="statey"/>
```

```
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `interaction.associate.done` - This event is sent when the request has been initiated to associate the interaction with the SCXML session.
- `error.interaction.associate` - This event is sent when the request itself has failed for some reason.
- `interaction.deleted` and `interaction.notcontrolled` - These events indicate that the interaction has been associated with the originating session.
- `interaction.added` and `interaction.present` - These events indicate that the interaction is now associated with the new session.

Note: A current limitation is that using the association function with an incorrect 'sessionid' results in the event 'interaction.associate.done' being generated, but the interaction remains with the current session.

<accept>

This is the action that accepts and answers an interaction at the specific resource. This is used to answer a voice interaction or accept a multi-media-based interaction. It is equivalent to the `TAnswerCall` request in T-Server. **Note:** This action is only for voice interactions at this time.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the

Name	Required	Type	Default Value	Valid Values	Description
					event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
resource	true	value expression	none	Any valid string or Resource Object.	A value expression which returns the address that this interaction is to be accepted for. For details, on the format, see Addressing Resources. See SCXML Legal Data Values and Value Expressions

The following is an example:

```
<state id="do_accept">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:accept requestid="_data.reqid"/>
</onentry>
<transition event="interaction.accept.done" target="statex"/>
<transition event="error.interaction.accept" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `interaction.accept.done` - This event is sent when the request has been initiated to accept theinteraction by the resource.
- `error.interaction.accept` - This event is sent when the request itself has failed for some reason.
- `Interaction.partystatechanged` - This event is sent when the resource accepts or answers the interaction for processing.

<attach>

This action (as well as the 2 other related actions <detach> and <associate>) change the ownership relation between sessions and interactions. It is used to associate an **ownerless** interaction with current session. Attempting to apply this action to an interaction owned by another session will fail - taking the interaction from another session is supposed to be done by agreement with this session the (see <associate> action). For multimedia interactions <attach> will not pull an interaction - it must be already pulled to work. Typically, when using this action for multimedia, the <detach> would be used prior to <attach>.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated

Name	Required	Type	Default Value	Valid Values	Description
					with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
Interactionid	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the interaction id that is to be attached to this session. See SCXML Legal Data Values and Value Expressions for details.

Example

```
<state id="do_attach">
```

```
    <datamodel>
        <data id="reqid"/>
    </datamodel>
    <onentry>
        <ixn:attach requestid="_data.reqid" interactionid="_event.data.id"/>
    </onentry>
    <transition event="interaction.attach.done" target="statex"/>
    <transition event="error.interaction.attach" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `interaction.attach.done` - Success; will be accompanied with asynchronous events interaction.added and interaction.present.
- `error.interaction.attach` - This event is sent when the request has failed for some reason (for example - interaction does not exist or interaction is owned by another session).

<detach>

This action (as well as the 2 other related actions <attach> and <associate>) change the ownership relation between sessions and interactions. It is the opposite of <attach> and is used to dis-associate an interaction from the current session. A session that <detach>'s for some interaction is still responsible for this interaction, or ensuring that the interaction is handled by some other session. This applies to both voice and multimedia interactions (in the case of multimedia, a detached interaction will not be returned back into queue). Normally after <detach> the session should either move the interaction to another session via <associate> or <attach> back to the interaction. Note. As <detach> action relies on interaction attached data it cannot be done immediately after assigning the interaction to the session. If invoked before interaction data properly updated the action will result error.interaction.detach with error invalidstate. In such cases application should repeat the action after short delay

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression

Name	Required	Type	Default Value	Valid Values	Description
					evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
Interactionid	true	value expression	none	Any value expression that	A value expression which

Name	Required	Type	Default Value	Valid Values	Description
				returns a valid string	returns the interaction id that is to be attached to this session. See SCXML Legal Data Values and Value Expressions for details.

Children

None

Events

The following events can be generated as part of this action:

- `interaction.detach.done` - Success; will be accompanied with asynchronous events interaction.notcontrolled and interaction.deleted.
- `error.interaction.detach` - This event is sent when the request has failed for some reason (for example - interaction does not exist or interaction is owned by another session).
- `interaction.notcontrolled` - This event is sent when the interaction is no longer associated with a session.
- `interaction.deleted` - This event is sent when the interaction is deleted from the current session.

Note: SCXML application may have transition for events interaction.notcontrolled and interaction.deleted in some of outer states. In this case, if those events, raised after ixn:detach action, will not be handled in a state where ixn:detach has been executed, they may cause undesired transitions. To avoid that, add target-less transitions for those events into state where ixn:detach is executed. In Composer, it may be done by adding exceptions for those events in "properties" of corresponding block (for example "Detach" or "ForceRoute" with detach=true), with "Target" checkbox un-checked.

Voice Interaction Actions

<createcall>

This is the initiation or creation of a voice interaction between a customer or resource and a resource. This is equivalent to the T-Server `TMakeCall` or `TMakePredictiveCall` functions.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
from	true	value expression	none	Any valid string or Resource Object	A value expression which returns the address that this call is to be made from. See SCXML Legal Data Values and Value Expressions
to	true	value expression	none	Any valid string or Resource Object	A value expression which returns the destination address for this call. See SCXML Legal Data Values and Value Expressions

Name	Required	Type	Default Value	Valid Values	Description
type	false	NMTOKEN	regular	regular, agent, supervisor, priority, agentpriority, predictive	This defines the type of call that is to be created. • Regular - is a call to a customer. • Agent - is a call to another agent. • Supervisor - is a call to a supervisor. • Priority - is a priority call to a customer. • Agentpriority - is a priority call to another agent. • Predictive - is a predictive call to a customer.
reason	false	value expression	none	Any valid string	A value expression which returns the reason for making this call. See SCXML Legal Data Values and Value Expressions for details.
udata	false	value expression	none	Any valid ECMAScript object	A value expression which returns a valid ECMAScript object. This object's data will become part of the call's interaction object. See SCXML Legal Data Values and Value Expressions for details.

Name	Required	Type	Default Value	Valid Values	Description
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when establishing this call. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details.
timeout	false	value expression	none	A value expression which returns an integer	A value expression which returns an integer which represents the number of seconds to wait. See SCXML Legal Data Values and Value Expressions for details. The integer returned is interpreted as a time interval. This interval begins when `<createcall>` is executed. The `<createcall>` must fail if not completed by the end of this interval. Completion is defined as the call getting to a CONNECTED state as signaled by an interaction.created event. A failed `<createcall>` must return either error.voice.createcall event or the interaction.partystatechanged event. This attribute is only used when the type attribute is set to "predictive".

The following is an example:

```
<state id="do_createcall">
  <datamodel>
        <data id="reqid"/>
  </datamodel>
  <onentry>
        <ixn:createcall requestid="_data.reqid" from="'1234'" to="'+1919466600'" />
  </onentry>
  <transition event="voice.createcall.done" target="statex"/>
  <transition event="error.voice.createcall" target="statey"/>
</state>
```

hints Attribute Considerations When property type of `<createcall>` action is set to predictive, property **extensions** of **hints** object allows to specify the content of `AttributeExtension` for the corresponding `TMakePredictiveCall` request. This is used to control call progress detection (CPD) processing. A valid value for property **extensions** is an ECMAScript object. Valid extensions to use for CPD purposes are described in the Deployment Guide for a specific T-Server. The following is an example using CPD to detect answering machine:

```
<state id="do_createcall">
  <datamodel>
        <data id="reqid"/>
  </datamodel>
  <onentry>
        <script>
                var cpdExtensions = {"extensions":{
                                        "cpd-record":"off",

"call_answer_type_recognition":"positive_am_detection",
                                        "cpd-on-connect":"off",
                                        "call_timeguard_timeout":"10000"}};
        </script>
        <ixn:createcall requestid="_data.reqid" from="'1234'" to="'+1919466600'"
type="predictive" hints="cpdExtensions" />
  </onentry>
  <transition event="voice.createcall.done" target="statex"/>
  <transition event="error.voice.createcall" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `voice.createcall.done` - This event is sent when the request has been accepted by the system and the call has started the creation process.
- `interaction.added` - This event is sent when either party is connected to the call.
- `interaction.partystatechanged` - This event is sent when the call has been created and either party is connected to the call.
- `interaction.partystatechanged` - This event is sent when the call has failed for some reason. This includes timeouts based on the timeout attribute.
- `error.voice.createcall` - This event is sent when the request itself has failed for some reason.

<hold>

This action puts a voice interaction on hold for a specific resource's device. This is equivalent to the T-Server `THoldCall` function.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
resource	true	value expression	none	Any valid string or Resource Object	A value expression which returns the

Name	Required	Type	Default Value	Valid Values	Description
					holding device address for this call. See SCXML Legal Data Values and Value Expressions

The following is an example:

```
<state id="do_holdcall">
<datamodel>
        <data id="reqid"/>
        <data id="ixnid"/>
</datamodel>
<onentry>
        <ixn:hold requestid="_data.reqid" interactionid="_data.ixnid" resource="'1234'" />
</onentry>
<transition event="voice.hold.done" target="statex"/>
<transition event="error.voice.hold" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `voice.hold.done` - This event is sent when the request has been accepted by the system and the interaction has started the hold process.
- `interaction.partystatuschanged` - This event is sent when the actual resource is on hold for the call.
- `<samp>error.voice.hold</samp>` - This event is sent when the request itself has failed for some reason.

<retrieve>

This action retrieves a voice interaction from hold for a specific resource's device. This is equivalent to the T-Server `TRetrieveCall` function.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data

Name	Required	Type	Default Value	Valid Values	Description
					model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
resource	true	value expression	none	Any valid string or Resource Object	A value expression which returns the retrieved device address for this call. See SCXML Legal Data Values and Value Expressions

The following is an example:

```
<state id="do_retrievecall">
<datamodel>
        <data id="reqid"/>
        <data id="ixnid"/>
</datamodel>
```

```
<onentry>
        <ixn:retrieve requestid="_data.reqid" interactionid="_data.ixnid" resource="'1234'" />
</onentry>
<transition event="voice.retrieve.done" target="statex"/>
<transition event="error.voice.retrieve" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `voice.retrieve.done` - This event is sent when the request has been accepted by the system and the interaction has started the retrieve process.
- `interaction.partystatuschanged` - This event is sent when the actual resource is retrieved from hold for the call.
- `error.voice.retrieve` - This event is sent when the request itself has failed for some reason.

<consultation>

This action extends an existing voice interaction to consult with a new resource's device. This is equivalent to the T-Server `TInitiateTransfer` and `TInitiateConference` functions.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event

Name	Required	Type	Default Value	Valid Values	Description
					handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
from	true	value expression	none	Any valid string or Resource Object	A value expression which returns the address that this consultation is to be made from. See SCXML Legal Data Values and Value Expressions
to	true	value expression	none	Any valid string or Resource Object	A value expression which returns the destination address for this consultation See SCXML Legal Data Values and Value Expressions
udata	false	value expression	none	Any valid ECMAScript object	A value expression which returns a valid ECMAScript object. This object's data will be come part of the consultation call's interaction object. See SCXML Legal Data Values and Value Expressions for details.

The following is an example:

```
<state id="do_consultationcall">
```

```
<datamodel>
        </datamodel>
<onentry>
        <ixn:consultation requestid="_data.reqid" interactionid="_data.ixnid" from="'1234'"
to="'5678'" />
</onentry>
<transition event="voice.consultation.done" target="statex"/>
<transition event="error.voice.consultation" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `voice.consultation.done` - This event is sent when the request has been accepted by the system and it has started the consultation process.
- `error.voice.consultation` - This event is sent when the request itself has failed for some reason.
- `interaction.partyadded` - This event is sent when the consulting or consulted party has been added to the consultation interaction.
- `interaction.partystatechanged` - This event is sent when the consulting or consulted party has changed states.

<alternate>

This action alternates a resource from a held call and an active call. This is equivalent to the T-Server `TAlternateCall` function.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this

Name	Required	Type	Default Value	Valid Values	Description
					attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
heldinteractionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
heldresource	true	value expression	none	Any valid string or Resource Object	A value expression which returns the address that is the held call. For details, see Addressing Resources. See SCXML Legal Data Values and Value Expressions
activeinteractionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
activeresource	true	value expression	none	Any valid string or Resource Object	A value expression which returns the address that is in the active call. For details, see Addressing Resources. See SCXML Legal Data

Name	Required	Type	Default Value	Valid Values	Description
					Values and Value Expressions

The following is an example:

```
<state id="do_alternatecall">
<datamodel>
        <data id="reqid"/>
        <data id="Aixnid"/>
        <data id="Hixnid"/>
</datamodel>
<onentry>
        <ixn:alternate requestid="_data.reqid" heldinteractionid="_data.Hixnid"
                heldresource="'1234'" activeinteractionid="_data.Aixnid"
activeresource="'5678'" />
</onentry>
<transition event="voice.alternate.done" target="statex"/>
<transition event="error.voice.alternate" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `voice.alternate.done` - This event is sent when the request has been accepted and the system has started the alternate process.
- `error.voice.alternate` - This event is sent when the request itself has failed for some reason.
- `interaction.partystatechanged` - This event is sent when the held and active resource in the calls changes state (held to active and active to held).

<reconnect>

This action drops the active call in the consultation and retrieves the held call for a specific resource. This is equivalent to the T-Server `TReconnectCall` function.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as

Name	Required	Type	Default Value	Valid Values	Description
					part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
heldinteractionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
heldresource	true	value expression	none	Any valid string orResource Object	A value expression which returns the address that is the held call. For details, see Addressing Resources. See SCXML Legal Data Values and Value Expressions
activeinteractionid	true	value expression		Any value expression that returns a valid	A value expression which returns the `_genesys.ixn.interactions[x].g_u`

Name	Required	Type	Default Value	Valid Values	Description
				string	associated with this request. See SCXML Legal Data Values and Value Expressions for details.
activeresource	true	value expression	none	Any valid string or Resource Object	A value expression which returns the address that is in the active call. For details, see Addressing Resources. See SCXML Legal Data Values and Value Expressions

The following is an example:

```
<state id="do_reconnectcall">
<datamodel>
        <data id="reqid"/>
        <data id="Aixnid"/>
        <data id="Hixnid"/>
</datamodel>
<onentry>
        <ixn:reconnect requestid="_data.reqid" heldinteractionid="_data.Hixnid"
heldresource="'1234'"
                activeinteractionid="_data.Aixnid" activeresource="'5678'" />
</onentry>
<transition event="voice.reconnect.done" target="statex"/>
<transition event="error.voice.reconnect" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `voice.reconnect.done` - This event is sent when the request has been accepted by the system, which has started the reconnect process.
- `interaction.partystatuschanged` - This event is sent when the held resource in the calls changes state.
- `interaction.partydeleted` - This event is sent when the parties of the active call are dropped.
- `interaction.deleted` - This event is sent when the active call is terminated.
- `error.voice.reconnect` - This event is sent when the request itself has failed for some reason.

<conference>

This action conferences a consultation call from a specific resource. This is equivalent to the T-Server `TCompleteConference` function.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
heldinteractionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
activeinteractionid	true	value expression		Any value expression that	A value expression which returns the

Name	Required	Type	Default Value	Valid Values	Description
				returns a valid string	`_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
resource	true	value expression	none	Any valid string or Resource Object	A value expression which returns the address that is conferencing the call. For details, see Addressing Resources. See SCXML Legal Data Values and Value Expressions

The following is an example:

```
<state id="do_conferencecall">
<datamodel>
        <data id="reqid"/>
        <data id="Aixnid"/>
        <data id="Hixnid"/>
</datamodel>
<onentry>
        <ixn:conference requestid="_data.reqid" heldinteractionid="_data.Hixnid"
                activeinteractionid="_data.Aixnid" resource="'1234'"/>
</onentry>
<transition event="voice.conference.done" target="statex"/>
<transition event="error.voice.conference" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `voice.conference.done` - This event is sent when the request has been accepted by the system, which has started the conference process.
- `error.voice.conference` - This event is sent when the request itself has failed for some reason.
- `interaction.onmerge` - This event is sent when the active call is merged with the held call.
- `interaction.partydeleted` - This event is sent when the party on the active call is dropped.
- `interaction.deleted` - This event is sent when the active call is terminated.

<transfer>

This action transfers a consultation call from a specific resource. This is equivalent to the T-Server `TCompleteTransfer` function.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
heldinteractionid	true	value expression		Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
activeinteractionid	true	value expression		Any value expression that	A value expression which returns the

Name	Required	Type	Default Value	Valid Values	Description
				returns a valid string	`_genesys.ixn.interactions[x].g_u` associated with this request. See SCXML Legal Data Values and Value Expressions for details.
resource	true	value expression	none	Any valid string or Resource Object	A value expression which returns the address that is transferring the call. For details, see Addressing Resources section. See SCXML Legal Data Values and Value Expressions

The following is an example:

```
<state id="do_transfercall">
<datamodel>
        <data id="reqid"/>
        <data id="Aixnid"/>
        <data id="Hixnid"/>
</datamodel>
<onentry>
        <ixn:transfer requestid="_data.reqid" heldinteractionid="_data.Hixnid"
                activeinteractionid="_data.Aixnid" resource="'1234'"/>
</onentry>
<transition event="voice.transfer.done" target="statex"/>
<transition event="error.voice.transfer" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `voice.transfer.done` - This event is sent when the request has been accepted by the system, which has started the transfer process.
- `error.voice.transfer` - This event is sent when the request itself has failed for some reason.
- `interaction.onmerge` - This event is sent when the held call and consult call on the transferring party are merged.
- `interaction.partydeleted` - This event is sent when the transferring party drops.
- `interaction.deleted` - This event is sent when the consult call is terminated.

<privateservice>

This action enables an application to pass data and request services (such as Set Feature, SIP Advice of Charge, change T-Server behavior, and so on) that are supported only by certain T-Servers and which are not covered by general feature requests. This is equivalent to the T-Server `TPrivateService` function and the applicable T-Server documentation should be consulted for the applicability of this request.

Attribute Details

Name	Required	Version	Type	Default Value	Valid Values	Description
requestid	false	8.1.1	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
serviceid	true	8.1.1	value expression	none	Any value expression that returns a valid integer	A value expression which returns an integer to indicate the

Name	Required	Version	Type	Default Value	Valid Values	Description
						type of information being passed or the service being requested. This is specific to the T-Server handling the call. Please refer to the T-Server documentation for your switch when setting this value. See SCXML Legal Data Values and Value Expressions for details.
interactionid	true	8.1.1	value expression	none	Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].` associated with this request. Non voice interactions will result in an error.voice.privateservice being generated. See SCXML Legal Data Values and Value Expressions for details.
resource	false	8.1.1	value expression	none	Any valid string or Resource Object	A value expression which returns the DN of the controlling agent or route point on whose behalf the information is provided. For details, see Addressing Resources. See SCXML Legal Data Values and Value Expressions **Note:** This corresponds to the thisDN parameter within the TLib

Name	Required	Version	Type	Default Value	Valid Values	Description
						TPrivateService method. Please refer to the T-Server documentation for your switch when setting this value
udata	false	8.1.1	ECMAScript Object	none	Any valid ECMAScript object	An ECMAScript Object which contains the list of key/value pairs which should be attached to the call in question. See SCXML Legal Data Values and Value Expressions for details.
reasons	false	8.1.1	ECMAScript Object	none	Any valid ECMAScript object	An ECMAScript Object which contains the list of key/value pairs which provide additional information associated with this private service request intended to specify reasons for and results of actions taken by the user. See SCXML Legal Data Values and Value Expressions for details.
extensions	false	8.1.1	ECMAScript Object	none	Any valid ECMAScript object	An ECMAScript Object which contains the list of key/value pairs which provides an additional data structure intended to take account of switch-specific features that cannot be described by other parameters or

Name	Required	Version	Type	Default Value	Valid Values	Description
						in the original structure of user data associated with this private service request. See SCXML Legal Data Values and Value Expressions for details.

Note: When submitting the Private Service request the target T-Server to which this request will be submitted will be determined based upon the following;

- If resource attribute contains both a switch and DN the switch will be used to locate the T-Server to submit the request to.
  - If resource attribute contains only a switch part, the switch will be used to locate the T-Server to submit the request to and the thisDN value of the underlying TLib TPrivateService will not be populated
- If resource attribute contains only a DN part, then the switch and associated T-Server will be determined from the accommodating interactionid. The switch will be determined based upon the first party that has the DN resource referenced.
- If resource is not provided then the T-Server will be determined by the last party within the associated parties entries for the supplied interaction. No resource (thisDN) will be provided in this case to the target T-Server.

In cases where by the target T-Server cannot be determined from the information provided then an error.voice.privateservice will be generated. The following is an example, please consult your T-Server manual for explicit information and appropriate examples on `TPrivateService` and applicable parameters:

```
<state id="do_private_service">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>

        <script>
                var myuserdata = { details : { name : "Smith, John", age : 45 } };
                var myreasons = { code : "New Update"};
                var myextensions = { keyname : "Its value" };
        </script>

        <ixn:privateservice requestid="_data.reqid"
        serviceid="1234" interactionid="_data.ixnid"
        resource="'9000'"
        udata = "myuserdata"
        reasons = "myreasons"
        extensions = "myextensions"/>
</onentry>

<transition event="voice.privateservice.done"
```

```
        cond="_event.data.requestid==_data.reqid"  target="statex"/>
<transition event="error.voice.privateservice"
        cond="_event.data.requestid==_data.reqid"  target="statey"/>

</state>
```

Children

None

Events

The following events can be generated as part of this action, please refer to your specific T-Server manual for details of how and when these events are to be generated as they are specific to the service and T-Server implementation:

- `voice.privateservice.done` - This event is sent when the request has been accepted by the orchestration system and sent. It is not an indication that the T-Server has handled or accepted the event.
- `error.voice.privateservice` - This event is sent when the request itself has failed for some reason.

<userevent>

This action provides access to the Tlib function TSendUserEvent and allows distributing of specified user events to registered clients of the TServer. ORS only supports sending of user events and cannot be the recipient of such events from other sources, to interact with sessions directly from an external source it is recommended to use the External Interfaces which are described here External Interfaces

Attribute Details

Name	Required	Version	Type	Default Value	Valid Values	Description
requestid	false	8.1.2	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not

Name	Required	Version	Type	Default Value	Valid Values	Description
						specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
resource	true	8.1.2	value expression	none	Any valid string or resource object	A value expression which returns the Switch and DN that will be used send message. The receving client for the user event should be this DN See SCXML Legal Data Values and Value Expressions for details.
event	false	8.1.2	value expression	none	Any valid ECMAScript object containing the zero one or more of the following properties: • AttributeCallID • AttributeCallType • AttributeCustomerID • AttributeConnID • AttributeThisDN • AttributeOtherDN • AttributeThirdPartyDN	An ECMAScript Object which contains the list of key/ value pairs which describe the content of sent event, with the exception of the user data, extensions and reasons which are provided if required in additional attributes

Name	Required	Version	Type	Default Value	Valid Values	Description
					• AttributeDNIS • AttributeANI • AttributeCollectedDigits • AttributeMediaType	for this action.
interactionid	false	8.1.2	value expression	none	interaction id	Id of interaction which properties will be partially used to populate attributes of distributed user event.
udata	false	8.1.2	ECMAScript Object	none	Any valid ECMAScript object	An ECMAScript Object which contains the list of key/value pairs which should be sent as user data with the user event See SCXML Legal Data Values and Value Expressions for details.
reasons	false	8.1.2	ECMAScript Object	none	Any valid ECMAScript object	An ECMAScript Object which contains the list of key/value pairs which should be sent as reason information with the user event See SCXML Legal Data Values and Value Expressions for details.
extensions	false	8.1.2	ECMAScript Object	none	Any valid ECMAScript object	An ECMAScript Object which contains the list of key/value pairs which should be sent as extensions with the user event. See SCXML Legal

Name	Required	Version	Type	Default Value	Valid Values	Description
						Data Values and Value Expressions for details.

Note 1. If the **interactionid** is provided in teh action then following 7 attributes will be copied from the specified interaction if they are not explcitely defined in **event** (and only them): AttributeCallID, AttributeCallType, AttributeCustomerID, AttributeConnID, AttributeDNIS, AttributeANI, AttributeCollectedDigits. **Note 2.** User Event Attributes type
AttributeCallID must be provided as string or number. If taken from interaction then matches to **voice.callid** property.

- **AttributeCallType** must be provided as string. If taken from interaction then matches to **voice.type** property.
- **AttributeCustomerID** must be provided as string. If taken from interaction then matches to **tenantid** property.
- **AttributeConnID** must be provided as string. If taken from interaction then matches to **voice.connid** property.
- **AttributeThisDN** must be provided as string. Cannot be taken from interaction.
- **AttributeOtherDN** must be provided as string. Cannot be taken from interaction.
- **AttributeThirdPartyDN** must be provided as string. Cannot be taken from interaction.
- **AttributeDNIS** must be provided as string. If taken from interaction then matches to **voice.dnis** property.
- **AttributeANI** must be provided as string. If taken from interaction then matches to **voice.ani** property.
- **AttributeCollectedDigits** must be provided as string. If taken from interaction then matches to **voice.ced** property.

- **AttributeMediaType** must be provided as number. Cannot be taken from interaction.

Sample:

```
<state id="do_userevent">
     <datamodel>
         <data id="reqid"/>
     </datamodel>
    <onentry>
       <script>
           var myuserdata = { "details": { "name": "Smith, John", "age": 45 } };
           var myreasons = { "code": "New Update" };
           var myextensions = { "keyname": "Its value" };
           var myevent = { "AttributeThisDN": "4000", "AttributeConnID":
genesys.ixn.interactions[0].voice.connid };
           var dest={ "switch":"target_switch", "dn" : "4000" };
        </script>
      <ixn:userevent requestid="_data.reqid"
          resource = "dest"
          event = "myevent"
          udata = "myuserdata"
          reasons = "myreasons"
          extensions = "myextensions"/>
     </onentry>
    <transition event="voice.userevent.done" cond="_event.data.requestid==_data.reqid"
target="statex"/>
    <transition event="error.voice.userevent" cond="_event.data.requestid==_data.reqid"
target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action :

- `voice.userevent.done` - This event is sent when the request has been accepted by the orchestration system and sent. It is not an indication that the T-Server has handled or accepted the event.
- `error.voice.userevent` - This event is sent when the request itself has failed for some reason.

<querycalls>

This action queries a call from a specific resource.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to

Name	Required	Type	Default Value	Valid Values	Description
					distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
resource	true	value expression	none	Any valid string or Resource Object	A value expression which returns the DN of the controlling agent or route point on whose behalf the information is provided. For details, see Addressing Resources. See SCXML Legal Data Values and Value Expressions

The following is an example:

```
<state id="do_querycalls">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:querycalls requestid="_data.reqid" resource="'600'" />
</onentry>

<transition event="voice.querycalls.done"
        cond="_event.data.requestid==_data.reqid"  target="statex"/>
<transition event="error.voice.querycalls"
        cond="_event.data.requestid==_data.reqid"  target="statey"/>

</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `voice.querycalls.done` - This event is sent when the request has been accepted by the orchestration system and sent.
- `error.voice.querycalls` - This event is sent when the request itself has failed for some reason.

Message Based Actions

The following are the msgbased-specific interaction actions.

<createmessage>

This action creates a new message and an associated interaction which can be used for the following purposes:

- Acknowledgements (source interaction - e-mail and open media)
- AutoResponses (source interaction - e-mail and open media)
- New e-mail notifications
- SMS messages (e-mail and native)
- Forwarded e-mail
- Reply e-mail from an external resource
- Redirect e-mail
- E-mail a chat transcript
- Create outbound e-mail

This action will also give the developer the options to send the message after it is created.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event

Name	Required	Type	Default Value	Valid Values	Description
					handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
type	false	NMTOKEN	emailout_reply	acknowledgement, auto_response, customer_reply, outbound_new, emailout_reply, inbound_collaboration_reply, ndr_interaction, outbound_collaboration_invite, outbound_notification, redirect, forwarded	This specifies the type of message header that is to be used for the message.
media	false	NMTOKEN	_genesys.ixn.mediaType.TMediaEMail	Values from the `_genesys.ixn.mediaType` enumeration object: TMediaAny, TMediaEMail, TMediaFax, TMediaNativeSMS, TMediaSMail, TMediaSMS, TMediaWebForm, TMediaOpenMedia	This is the media type of the message. The TMediaOpenMedia is only valid when the type attribute value is "acknowledgement" or "autoresponse".
server	false	value expression	If not supplied, the functional module will determine the server based on the type attribute	Any value expression that returns a valid string	A value expression which returns the name of the msgbased server to create this message on. See SCXML Legal Data Values and Value Expressions for details.
relatedixnid	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` of the interaction that is associated with the interaction being created with this request. There are special values that can be returned:

Name	Required	Type	Default Value	Valid Values	Description
					• **"$NOT"** - means that there is not an interaction that is to be related to the new one. See SCXML Legal Data Values and Value Expressions for details.
thread	false	boolean expression	true	Any boolean expression that returns a true or false	A boolean expression which returns a boolean that identifies whether this new message is going to be part of the related interaction's thread. See SCXML Conditional Expressions for details.
includeorig	false	boolean expression	false	Any boolean expression that returns a true or false	A boolean expression which returns a boolean that identifies whether the original message is going to be added to this new message. See SCXML Conditional Expressions for details.
msgsrc	false	value expression	none	Any of the following valid URI schemes: • gdata	A value expression which returns a URI that identifies the location of the source message (suggested response, and so on) to use for this new message. The following are the URI schemes that are supported: • gdata • Configuration

Name	Required	Type	Default Value	Valid Values	Description
					data (for example, gdata:config\CA.name or gdata:config\SR.name) - This will be the name (id) of the suggested responses or categories from the Configuration Server. • User data (for example, gdata:udata) - this indicates that the message will be taken from the user data of the related interaction. So the related interaction must have key/value pairs for the CategoryId key. If SRLid and

Name	Required	Type	Default Value	Valid Values	Description
					CtgId are in the related interaction's message and the CategoryId in udata is null, then an error is generated (error.msgbased.create See SCXML Legal Data Values and Value Expressions for details.
to	false	value expression	none	Any list of valid addresses. This will be a single-quoted string with the address URIs separated by either a "," or ";".	A value expression which returns a list of addresses to send this message to. An example is to='joe@cox.com;joe@coy.com' or to="_origin.all". The following are the valid values for this attribute: • _origin • _origin.all • _udata - if this URI value is used, then this can be the only entry in list. You can also specify a specific key name to be using `_udata\\<keyname>`. • List of addresses See SCXML Legal Data Values and Value Expressions for details.

Name	Required	Type	Default Value	Valid Values	Description
from	false	value expression	none	Any valid e-mail address(es). If multiple addresses are supplied, the string with the URIs will be single-quoted and will be separated by either a "," or ";".	A value expression which returns the address(es) that this message will be from. An example is from="joe@cox.com" or from="_origin". The following are the valid values for this attribute: • _origin • List of addresses **Note:** This attribute is required when the type attribute value is: • outbound_new See SCXML Legal Data Values and Value Expressions
cc	false	value expression	none	Any list of valid addresses. This will be a single-quoted string with the address URIs separated by either a "," or ";".	A value expression which returns a list of addresses to send a copy of this message to. An example is cc='joe@cox.com;joe@coy.com'. cc="_origin" The following are the valid values for this attribute: • _origin • List of addresses See SCXML Legal Data Values and Value Expressions
exclude	false	value expression	none	Any list of valid addresses. This will be a single-quoted string with the address URIs separated by either a "," or ";".	A value expression which returns a list of addresses to exclude from the "to" or "cc" list of addresses. An example is exclude='joe@cox.com;joe@coy.com'. The following are the valid values for this attribute:

Name	Required	Type	Default Value	Valid Values	Description
					• List of addresses See SCXML Legal Data Values and Value Expressions
subject	false	value expression	none	Any valid string	A value expression which returns the subject of the new message. There are special values that can be returned: • **"$USESRL"** means the functional module will use the subject from the suggested response template. If not specified, there will be no subject for the new message. See SCXML Legal Data Values and Value Expressions for details.
queue	false	value expression	none	Any valid string	A value expression which returns the queue that the new interaction or message should be put into when it is created. The SCXML session associated with this queue will be responsible for explicitly sending this message or interaction later.The following are the values that can be provided and what processing will be done: • **"$CURQUEUE"** means that the current

Name	Required	Type	Default Value	Valid Values	Description
					queue associated with the interaction is used. This value cannot be used if the relatedixnid value is "$NOT". • If no value is specified the message will be sent by platform automatically. The session will not get any other events (except masgbased.createmessage related with this message. It is recommended that you do not use this attribute and allow the functional module to send the interaction or message. See SCXML Legal Data Values and Value Expressions for details.
chattranscript	false	boolean expression	false	Any valid boolean value	A boolean expression which identifies whether the chat transcript of the related interaction (must be a chat interaction) should be attached to the message. This attribute is only valid when the relatedixn attribute does not have a value of "NOT". See SCXML

Name	Required	Type	Default Value	Valid Values	Description
					Conditional Expressions for details.
delivery	false	boolean expression	false	Any boolean expression that returns true or false	A boolean expression which returns a boolean that identifies whether this message being sent should include a request for a return message indicating whether and how the original message was delivered. • If one of the SMTP servers involved in the transport of the original message fails to deliver it, the return message comes into the system with subtype InboundNDR. It contains no additional information. • If the original message is successfully delivered, the return message comes into the system with subtype InboundReport. It uses attached data to

Name	Required	Type	Default Value	Valid Values	Description
					indicate delivery statuses such as delayed, delivered, relayed, and so on. This attribute is only used when the "send" attribute is equal to "true". User's Guide. See SCXML Conditional Expressions for details.
disposition	false	boolean expression	false	Any boolean expression that returns true or false	A boolean expression which returns a boolean that identifies whether the message should include a request for a return message indicating what happened to the original message after it was delivered; for example, whether it was displayed, printed, deleted without displaying, and so on. The return message comes into the system with subtype InboundDisposition and it provides the delivery status in attached data. This attribute is only used when the "send" attribute is equal to "true". See SCXML Conditional Expressions for details.
codefields	false	ECMAScript object	none	Any valid ECMAScript object with a set of key/ value properties	An ECMAScript Object which contains the list of key/value pairs which should be used to substitute the values into the

Name	Required	Type	Default Value	Valid Values	Description
					suggested response message that is created. See SCXML Legal Data Values and Value Expressions for details.
associated	false	boolean expression	true	Any valid boolean value	Release 8.1.400.36 introduces an optional Boolean attribute, associated, for the <ixn:createmessage> action. The default value is true to comply with previous behavior. If the associated attribute is set to false, a new interaction is not associated with the session that created it. In this case, the creating session should not expect any asynchronous events (such as interaction.added, and so on) that are related to the new interaction and the only published event will be msgbased.createmessage.done.

The following is an example:

```
<state id="do_simple_autorsp_or_Ack">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:createmessage requestid="_data.reqid" type="customer_reply"
server="'JEmailServer1'"
                        msgsrc="'gdata:config\CA.SR27'" includeorig="true" to="'_origin.all'"
                        from="'_origin'" subject="'$USESRL'">
                <ixn:field key="'$servicename'" value="_data.service"/>
        </ixn:createmessage>
</onentry>
<transition event="msgbased.createmessage.done" target="statex"/>
<transition event="error.msgbased.createmessage" target="statey"/>
</state>
<state id="do_new_email">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:createmessage requestid="_data.reqid" type="emailout_reply"
```

```
server="'JEmailServer1'"
                        msgsrc="'gdata:config\CA.SR45'" relatedixn="'$NOT'"
to="'joee@abc.com'"
                        from="'doitco@xyz.com'" subject="'$USESRL'">
                <ixn:field key="'$servicename'" value="_data.service"/>
        </ixn:createmessage>
</onentry>
<transition event="msgbased.createmessage.done" target="statex"/>
<transition event="error.msgbased.createmessage" target="statey"/>
</state>
<state id="do_new_sms">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:createmessage requestid="_data.reqid" type="outbound_notification"
server="'JEmailServer1'"
                        msgsrc="'gdata:config\CA.SR888'" relatedixn="'$NOT'"
to="'joee@abc.com'"
                        from="'doitco@xyz.com'" subject="'Hello to Bank xyz'">
                <ixn:field key="'$custname'" value="_data.custname"/>
        </ixn:createmessage>
</onentry>
<transition event="msgbased.createmessage.done" target="statex"/>
<transition event="error.msgbased.createmessage" target="statey"/>
</state>
<state id="do_forward">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:createmessage requestid="_data.reqid" type="forwarded" server="'JEmailServer1'"
                thread="false" to="'joee@abc.com'" from="'doitco@xyz.com'"/>
</onentry>
<transition event="msgbased.createmessage.done" target="statex"/>
<transition event="error.msgbased.createmessage" target="statey"/>
</state>
<state id="do_redirect">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:createmessage requestid="_data.reqid" type="redirect" server="'JEmailServer1'"
                thread="false" to="'joee@abc.com'" from="'doitco@xyz.com'"/>
</onentry>
<transition event="msgbased.createmessage.done" target="statex"/>
<transition event="error.msgbased.createmessage" target="statey"/>
</state>
<state id="do_reply_from_extresource">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:createmessage requestid="_data.reqid" type="inbound_collaboration_reply"
                server="'JEmailServer1'" thread="false" to="'joee@abc.com'"
                from="'doitco@xyz.com'"/>
</onentry>
<transition event="msgbased.createmessage.done" target="statex"/>
<transition event="error.msgbased.createmessage" target="statey"/>
</state>
<state id="do_chat_transcript">
<datamodel>
        <data id="reqid"/>
```

```
</datamodel>
<onentry>
        <ixn:createmessage requestid="_data.reqid" type="emailout_reply"
                server="'JEmailServer1'" msgsrc="'gdata:config\CA.SR988'"
                thread="false" relatedixnid="_genesys.ixn.interactions[x].g_uid"
                to="'joee@abc.com'" from="'doitco@xyz.com'" chattranscript="true"/>
</onentry>
<transition event="msgbased.createmessage.done" target="statex"/>
<transition event="error.msgbased.createmessage" target="statey"/>
</state>
<state id="do_new_task">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:createmessage requestid="_data.reqid" type="outbound_new"
                server="'JEmailServer1'" msgsrc="'gdata:config\CA.SR999'"
                thread="false" relatedixnid="'$NOT'" to="'agent1'"
                from="'doitco@xyz.com'"/>
</onentry>
<transition event="msgbased.createmessage.done" target="statex"/>
<transition event="error.msgbased.createmessage" target="statey"/>
</state>
```

Required Attributes based on Request and Media Type

Note: For `<createmessage>`, all request types except outbound_new require a related interaction (relatedixnid). For send, all requests require a related interaction (relatedixnid). The related interaction id defaults to `_genesys.ixn.interactions[main_ixnid].g_uid` if none is specified on the request (so it's not listed as a required attribute in the following table).

Type	Media	Required Attributes	Comments
acknowledgement, auto_response	TMediaEMail, TMediaOpenMedia	msgsrc	msgsrc may be gdata:config\CA.id or gdata:config\SA.id or gdata:udata A valid related interaction is required.
acknowledgement, auto_response	TMediaNativeSMS	msgsrc	msgsrc contains text string of message. A valid related interaction is required.
forwarded	TMediaEMail	msgsrc, to	msgsrc must be gdata:config\SA.id A valid related interaction is required.
inbound_collaboration_reply	TMediaEMail		no required attributes A valid related interaction is required. The related interaction must be of type "InboundCollaborationReply"
outbound_new	TMediaEMail, TMediaSMS	msgsrc, from	If 'to' is not specified then 'ContactId' must be in user

Type	Media	Required Attributes	Comments
			data of related interaction. msgsrc may be gdata:config\CA.id or gdata:config\SA.id or gdata:udata
outbound_new	TMediaNativeSMS	msgsrc, to, from	msgsrc contains text string of message.
outbound_notification	TMediaEMail	msgsrc	A valid related interaction is required.
redirect	TMediaEMail	to	A valid related interaction is required.

Children

None

Events

The following events can be generated as part of this action:

- `msgbased.createmessage.done` - This event is sent when the request has been accepted by the system, creating the msgbased interaction.
- `error.msgbased.createmessage` - This event is sent when the request has failed for some reason.
- `interaction.present` - This event is sent when the actual new msgbased interaction is created and is available to the session (if the queue is not specified).

<sendmessage>

This action sends a message that was created either by the `<createmessage>` action or by an outside source (for example, agent desktop).

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location

Name	Required	Type	Default Value	Valid Values	Description
					Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
server	false	value expression	If not supplied, the functional module will determine the server based on the media type of the interaction	Any value expression that returns a valid string	A value expression which returns the name of the msgbased server to create this message on. See SCXML Legal Data Values and Value Expressions for details.
interactionid	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_u` of the interaction that is associated with this request. There is a special value that can be returned: • ECMAScript **Null** means the functional module will not use an interaction for the request.

Name	Required	Type	Default Value	Valid Values	Description
					See SCXML Legal Data Values and Value Expressions for details.
delivery	false	boolean expression	false	Any boolean expression that returns a true or false	A boolean expression which returns a boolean which identifies if the message being sent should include a request for a return message indicating whether and how the original message was delivered. • If one of the SMTP servers involved in the transport of the original message fails to deliver it, the return message comes into the system with subtype InboundNDR. It contains no additional information. • If the original message is successfully delivered, the return message comes into the system with subtype InboundReport. It uses attached data to

Name	Required	Type	Default Value	Valid Values	Description
					indicate delivery statuses such as delayed, delivered, relayed, and so on. See SCXML Conditional Expressions for details.
disposition	false	boolean expression	false	Any boolean expression that returns a true or false	A boolean expression which returns a boolean that identifies whether the message should include a request for a return message indicating what happened to the original message after it was delivered; for example, whether it was displayed, printed, deleted without displaying, and so on. The return message comes into the system with subtype InboundDisposition, and it provides the delivery status in attached data. See SCXML Conditional Expressions for details.
from	false	value expression	none	A valid address URI	A value expression which returns the address that this message will be from. If specified, it will overwrite the current "from" address in the message. An example is from="joe@cox.com" The following are the valid values for this attribute: • address

Name	Required	Type	Default Value	Valid Values	Description
					See SCXML Legal Data Values and Value Expressions
cc	false	value expression	none	Any list of valid addresses. This will be a single-quoted string with the address URIs separated by either a "," or ";".	A value expression which returns a list of additional addresses to send a copy of this message to. These addresses are added to the current list of carbon copy addresses in the message being sent. An example is cc='joe@cox.com;joe@coy.com'. The following are the valid values for this attribute: • List of addresses See SCXML Legal Data Values and Value Expressions
exclude	false	value expression	none	Any list of valid addresses. This will be a single-quoted string with the address URIs separated by either a "," or ";".	A value expression which returns a list of additional addresses to exclude from the "to" or "cc" list of addresses. An example is exclude='joe@cox.com;joe@coy.com' The following are the valid values for this attribute: • List of addresses See SCXML Legal Data Values and Value Expressions
subject	false	value expression	none	Any valid string	A value expression which returns the subject of the message that is to be sent. This will overwrite the subject currently assigned to the message. See SCXML Legal Data

Name	Required	Type	Default Value	Valid Values	Description
					Values and Value Expressions for details.
headerfields	false	ECMAScript object	none	Any valid ECMAScript object with a set of key/value properties.	An ECMAScript Object which contains the list of key/value pairs which should be used to add headers to the message when it is sent. See SCXML Legal Data Values and Value Expressions for details.

The following is an example:

```
<state id="do_send">
<datamodel>
        <data id="reqid"/>
</datamodel>
<onentry>
        <ixn:sendmessage requestid="_data.reqid" server="'JEmailServer1'"
                delivery="true"/>
</onentry>
<transition event="msgbased.sendmessage.done" target="statex"/>
<transition event="error.msgbased.sendmessage" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `msgbased.sendmessage.done` - This event is sent when the request has been accepted by the system and the msgbased message is being sent.
- `error.msgbased.sendmessage` - This event is sent when the request has failed for some reason.

<getcontent >

This action gets the latest version of the message and updates the `_genesys.ixn.interactions[].msgbased.content` property

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_` of the interaction that is associated with this request. There is a special value that can be returned: • ECMAScript **Null** means the functional module will not use an interaction for the

Name	Required	Type	Default Value	Valid Values	Description
					request. See SCXML Legal Data Values and Value Expressions for details.

The following is an example:

```
<state id="do_getContent">
<datamodel>
        <data id="reqid"/>
        <data id="ixnid"/>
        <data id="currentixn"/>
</datamodel>
<onentry>
        <ixn:getcontent requestid="_data.reqid"
                interactionid="'_genesys.ixn.interactions[_data.ixnid].g_uid"/>
</onentry>
<transition event="msgbased.getcontent.done" target="statex"/>
<transition event="error.msgbased.getcontent" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- msgbased.getcontent.done - This event is sent when the request has been accepted by the system and the content has been refreshed.
- error.msgbased.getcontent - This event is sent when the request has failed for some reason.

Chat Actions

The following are the chat-specific interaction actions.

<gettranscript >

This action gets the latest version of the chat transcript and returns it in the chat.gettranscript.done event.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
interactionid	false	value expression	none	Any value expression that returns a valid string	A value expression which returns the `_genesys.ixn.interactions[x].g_` of the interaction that is associated with this request. There is a special value that can be returned: • ECMAScript **Null** means the functional module will not use an interaction for the

Name	Required	Type	Default Value	Valid Values	Description
					request. See SCXML Legal Data Values and Value Expressions for details.

The following is an example:

```
<state id="do_getTranscript">
<datamodel>
        <data id="reqid"/>
        <data id="currentixn"/>
</datamodel>
<onentry>
        <ixn:gettranscript requestid="_data.reqid"
                interactionid="'_genesys.ixn.interactions[1].g_uid"/>
</onentry>
<transition event="chat.gettranscript.done" target="statex"/>
<transition event="error.chat.gettranscript" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `chat.gettranscript.done` - This event is sent when the request has been accepted by the system and the transcript has been refreshed.
- `error.chat.gettranscript` - This event is sent when the request has failed for some reason.

Interaction Interface Events

Interaction Events

The following are the common events across interactions:

Event	Attributes	Description
interaction.terminate.done		This event indicates the success of the request and that the interaction is being terminated.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction id that was terminated.
error.interaction.terminate		This indicates that an abnormal condition occurred while trying to perform this request. This event will be sent as a result of a timeout of the request as well or due to problems with the request itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • unknown • invalidstate.state (null, ringing, hold, transferring, treating, routed) • remote
	description	This is a more detailed description of the error.
interaction.redirect.done		This event indicates the success of the request and that the interaction is being redirected.
	requestid	This is the ID associated with the request.

Event	Attributes	Description
	interactionid	This is the interaction id associated with the request.
error.interaction.redirect		This indicates that an abnormal condition occurred while trying to perform this request. This event will be sent as a result of a timeout of the request as well as due to problems with the request itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • unknown • invalidstate.state (null, ringing, hold, transferring, treating, routed) • invaliddestination • badtranslation • remote
	description	This is a more detailed description of the error.
interaction.associate.done		This event indicates the success of the request and that the interaction is now associated the SCXML session.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the request.
error.interaction.associate		This indicates that an abnormal condition occurred while trying to perform this request. This event will be sent as a result of a timeout of the request as well as due to problems with the request itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values:

Event	Attributes	Description
		• unknown • invalidserver • nointeraction
	description	This is a more detailed description of the error.
interaction.attach.done		This event indicates the success of the request and that the interaction is now associated the SCXML session.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the request.
error.interaction.attach		This indicates that an abnormal condition occurred while trying to perform this request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • unknown • nointeraction
	description	This is a more detailed description of the error.
interaction.detach.done		This event indicates the success of the request and that the interaction now is not associated with any SCXML session.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the request.
error.interaction.detach		This indicates that an abnormal condition occurred while trying to perform this request.

Event	Attributes	Description
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • unknown • nointeraction • invalidstate
	description	This is a more detailed description of the error.
interaction.accept.done		This event indicates the success of the request and that the interaction is being picked.
	requestid	This is the ID associated with the request.
error.interaction.accept		This indicates that an abnormal condition occurred while trying to perform this request. This event will be sent as a result of a timeout of the request as well as due to problems with the request itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • unknown • invalidstate.state (null, hold, transferring, treating, routed) • reject.reason (reason is the reason why the destination reject it.) • remote
	description	This is a more detailed description of the error.
interaction.clear.done		This event indicates the success of the request and that the resource's connection to the interaction is being cleared.

Event	Attributes	Description
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID where the resource's connection is cleared.
	resource	This is the resource's address ID of the resource's connection being cleared. [This is unclear.]
error.interaction.clear		This indicates that an abnormal condition occurred while trying to perform this request. This event will be sent as a result of a timeout of the request as well as due to problems with the request itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • unknown • invalidstate.state (null, ringing, hold, transferring, treating, routed) • remote
	description	This is a more detailed description of the error.
interaction.singlesteptransfer.done		This event indicates the success of the request and that the interaction has been transferred to the specified resource.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID which is being transferred.
error.interaction.singlesteptransfer		This indicates that an abnormal condition occurred while trying to perform this request. This event will be sent as a result of a timeout of the request as well as due to problems with the request itself.
	requestid	This is the ID associated with the request.

Event	Attributes	Description
	error	This is the type of error that occurred. The following are the possible values: <ul><li>unknown</li><li>invalidstate.state (null, ringing, hold, transferring, treating, routed)</li><li>remote</li></ul>
	description	This is a more detailed description of the error.
interaction.singlestepconference.done		This event indicates the success of the request and that the specified resource has been conferenced into the interaction.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID which has been conferenced.
error.interaction.singlestepconference		This indicates that an abnormal condition occurred while trying to perform this request. This event will be sent as a result of a timeout of the request as well as due to problems with the request itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: <ul><li>unknown</li><li>invalidstate.state (null, ringing, hold, transferring, treating, routed)</li><li>remote</li></ul>
	description	This is a more detailed description of the error.

Asynchronous Events

The following are the Interaction asynchronous events:

Event	Attributes	Description
interaction.added		This event indicates that a new interaction is associated with the session (it has been added to the _genesys.ixn.interactions[] array).Reasons for this event can vary - starting of session because of this interaction, some session activity resulted in the creation of new interaction, `<associate>` action and the target session, or `<createmessage>` action, and so on.
	interactionid	This is the interaction ID of the interaction that was added. Starting from that moment, the corresponding interaction object can be accessed, for example as _genesys.ixn.interactions[_event.data.interactionid]
interaction.deleted		This event indicates that the interaction is not associated any more with this session (it has been removed from the _genesys.ixn.interactions[] array). Reasons for this event can vary - interaction is finished or abandoned, `<associate>` action and the source session, and so on.Every time when interaction.deleted is going to be published, the platform will also publish an interaction.notcontrolled (see below) event if the last interaction.present event (see below) was not "notcontrolled" yet.
	interactionid	This is the interaction ID of the deleted interaction.Any attempt to access the _genesys.ixn.interactions[] array with the interaction ID of this interaction will result in a runtime error.
interaction.present		This event indicates that the interaction is under session control and the session is allowed to perform actions on the interaction, such as routing, redirecting, releasing, and so on. In the case of multi-media interactions, action or function invocations taken before getting the interaction.present event can result in runtime errors.In the case of voice interactions, action or function invocations can be taken after the interaction.added event, but they to have common processing for all interaction types. You should use the

Event	Attributes	Description
		interaction.present event for this. [?]
	interactionid	This is the interaction ID of the interaction that is available to the session for processing.
interaction.notcontrolled		This event accompanies interaction.present and means the session, although still owning the interaction, is no longer allowed to control it. Attempts to do so after this event will result in runtime errors.For voice interactions, this event will be published immediately before the interaction.deleted event.For multimedia interactions this event will be published when the Interaction Server moves (for any reason) the interaction from the platform back into the interaction queue.
	interactionid	This is the interaction ID of the interaction that is available to the session for processing.
	resultof	Starting with ORS 8.1.400.17 a new resultof property is added to the interaction.notcontrolled event. Possible values are: detaching, deletion, routing, revoking. • detaching - Interaction became notcontrolled because it was detached from running session. • deletion - Interaction became notcontrolled because it was deleted. • routing - Interaction became notcontrolled because of ORS actions like delivering to agent, placing in queue, and so on. So an event with resultof = routing is an expected event in a session because it is a result of some actions of strategy itself. • revoking - Interaction being processed by ORS (pulled) is taken away from that ORS instance explicitly (by agent, by media server, by Interaction Server due to

Event	Attributes	Description
		inactivity) or implicitly (disconnect from Interaction Server so all previously pulled interactions are not considered as such anymore). So an event with resultof = revoking is a kind of unexpected event in a session. Most probably the session should be terminated upon receiving such event, but this is up to the application developer. Unexpected Events: The most reliable way for an SCXML session to behave in the case of a primary Interaction Server failure is to terminate upon receiving an unexpected interaction.notcontrolled event. These unexpected interaction.notcontrolled events are the result of implicit revocation of all interactions due to an Interaction Server disconnect. This behavior is especially actual for an ORS cluster with several (more than one) nodes pulling interactions (especially with default Interaction Server configuration – delayed Udata updates). The first two values of resultof (detaching and deletion) are related to voice and multimedia. In this case, interaction.notcontrolled event with resultof =('detaching'|'deletion') is followed by an interaction.deleted event with the same resultof property. Values routing and revoking can be seen only in notcontrolled events for pure multimedia interactions. They are not related to voice or chat through SIPS.
interaction.partyadded		This event indicates that a new party has been created in an interaction. The interaction.partyadded event means adding a new party to the interaction.parties[] array. The event is published only if the platform detects that the party is being added to some interaction and that interaction is already associated with a session. So parties acquired by the interaction before the interaction.added event will not be reported through the interaction.partyadded event (added interactions will have them from the beginning). In the case of multimedia interactions, parties are agents only. Interaction queues, workbins, or the

Event	Attributes	Description
		interaction itself are not considered as parties and therefore the interaction.partyadded will not be generated when they are added to the interaction.
	interactionid	This is the interaction ID
	focusdeviceid	This is the device ID of the party
	partyid	This is the party ID
	partystate	This is the state of the party
interaction.partydeleted		This event indicates that a party has been deleted from an interaction. Just like interaction.deleted means removing an interaction from the _genesys.ixn.interactions[] array, interaction.paprtydeleted event means removing some party from the interaction.parties[] array.
	interactionid	This is the interaction ID
	focusdeviceid	This is the device ID of the party
	partyid	This is the party ID
	partystate	This is the state of the party
interaction.partystatechanged		This event indicates that state of the party in an interaction has been changed. The event is published only when the interaction is associated with a session and the platform detects a party state change for a party associated with the interaction.
	interactionid	This is the interaction ID that is being made available to the session.
	focusdeviceid	This is the device ID of the party that the interaction is being presented to.
	partyid	This is the party ID

Event	Attributes	Description
	partystate	This is the state of the party
	hints	This is functional module implementation-specific data that is associated with this event.
interaction.udata.changed		This event indicates that the user data associated with the interaction has been changed asynchronously (that is, changed by an outside source). Updated user data is available for the application in the data model, in the corresponding interaction object.
	interactionid	This is the ID of the interaction which has had the udata changed.
interaction.ondivert		This event indicates that interaction has been diverted. Note that after this event is signaled, events about new parties (new destinations) will also be sent to the SCXML application.
	hints	Starting with release 8.1.400.53, in the case of a voice interaction, the SCXML event interaction.ondivert will contain a `hints` attribute. The content of that attribute is functional module implementation-specific data that is associated with this event.
	interactionid	This is the ID of the interaction that is being made available to the session.
	divertingpartyid	This is the ID of the party that diverted the interaction.
	divertingresource	This is the device ID of the party that diverted the interaction.
	divertingpartystate	This is the state of the party that diverted the interaction.
	newdestination	This is the destination that the interaction has been diverted to.
interaction.onrouterequest (since 8.1.200.73)		This event indicates that the TEvent, EventRouteRequest, has been received for this interaction. Normally the

Event	Attributes	Description
		interaction.onrouterequest event can be ignored, except in the case that the EventRouteRequest is delayed from the EventQueued due to scripting in the switch, for instance by an Avaya VDN script which performs processing before asking that the call be routed. In these cases, the SCXML application can suspend processing by waiting for the interaction.onrouterequest event before proceeding with the session processing, or effectively, until the switch is ready for routing.
	interactionid	This is the ID of the interaction that is being made available to the session.
interaction.onsinglesteptransfer		This event indicates that a single step transfer has been performed in the interaction
	interactionid	This is the interaction ID that is being made available to the session.
	transferringpartyid	This is the ID of the party that transferred the interaction.
	transferringresource	This is the device ID of the party that transferred the interaction.
	transferringpartystate	This is the state of the party that transferred the interaction.
	newdestination	This is the transfer destination.
interaction.onsinglestepconference		This event indicates that a single step conference has been performed in the interaction.
	interactionid	This is the interaction ID that is being made available to the session.
	conferencingpartyid	This is the ID of the party that initiated the conference.
	conferencingresource	This is the device ID of the party that initiated the conference.

Event	Attributes	Description
	conferencingpartystate	This is the state of the party that transferred the interaction.
	addedpartyid	This is the ID of the party that has been added to the interaction.
	addedresource	This is the ID of the device that has been added to the interaction.
	addedpartystate	This is the state of the party that has been added to the interaction.
interaction.onmerge		This event indicates that two interactions were merged (indicating consult call completion). There are two types of merge possible: transfer completion or conference completion. The type of merge is provided in the hints in the Call Control Event.
	frominteractionid	This is the interaction ID that is being merged (consult).
	tointeractionid	This is the interaction ID that is the merge destination (primary).
	activeresource	This is the ID of the active device in the primary or consult call pair.
	activepartyid	This is the ID of the active party in the primary or consult call pair.
	activepartystate	This is the state of the active party in the primary or consult call pair.
	heldresource	This is the ID of the held device in the primary or consult call pair.
	heldpartyid	This is the ID of the held party in the primary or consult call pair.
	heldpartystate	This is the state of the held party in the primary or consult call pair.
interaction.property.changed (Since 8.1.400.39)		For multimedia interactions, this event will be published upon an

Event	Attributes	Description
		OnCallInfoChangedEx notification. Notifies ORS of a change in an interaction property. It also changes the interaction object property for the interaction referred to by the interactionid attribute.
	interactionid	This is the interaction ID that is being made available to the session.

hints Attribute Considerations The hints property of the interaction events is populated as follows:

- **hints.ccevent** - Call Control Event - This gives detailed definition of the interaction events being raised. This property is an integer and may have one of the following values:
  - 0 - CallControlEvent_Bridged
  - 1 - CallControlEvent_ConnectionCleared
  - 2 - CallControlEvent_Delivered
  - 3 - CallControlEvent_Established
  - 4 - CallControlEvent_Failed
  - 5 - CallControlEvent_Held
  - 6 - CallControlEvent_Offered
  - 7 - CallControlEvent_Originated
  - 8 - CallControlEvent_Queued
  - 9 - CallControlEvent_Retrieved
  - 10 - CallControlEvent_ServiceInitiated
  - 11 - CallControlEvent_Transfered
  - 12 - CallControlEvent_Conferenced
  - 13 - CallControlEvent_Diverted
- **hints.cause** - Call Control Event Cause - This is a detailed definition of the cause of the interaction events being raised. This property is an integer and may have one of the following values:
  - 0 - CCEventCause_unknown
  - 1 - CCEventCause_busy
  - 2 - CCEventCause_conference
  - 3 - CCEventCause_distributed
  - 4 - CCEventCause_distributionDelay
  - 5 - CCEventCause_enteringDistribution
  - 6 - CCEventCause_normal

  - 7 - CCEventCause_redirected
  - 8 - CCEventCause_singleStepConference
  - 9 - CCEventCause_singleStepTransfer
  - 10 - CCEventCause_transfer
- **hints.callstate (since 8.1.3)** - Call Control Event Call State - This is a detailed definition of the current call state of the interaction associated with the event being raised. This optional property is an integer and may have one of the following values:
  - 0 - CallStateOk
  - 1 - CallStateTransferred
  - 2 - CallStateConferenced
  - 3 - CallStateGeneralError
  - 4 - CallStateSystemError
  - 5 - CallStateRemoteRelease
  - 6 - CallStateBusy
  - 7 - CallStateNoAnswer
  - 8 - CallStateSitDetected
  - 9 - CallStateAnsweringMachineDetected
  - 10 - CallStateAllTrunksBusy
  - 11 - CallStateSitInvalidnum
  - 12 - CallStateSitVacant
  - 13 - CallStateSitIntercept
  - 14 - CallStateSitUnknown
  - 15 - CallStateSitNocircuit
  - 16 - CallStateSitReorder
  - 17 - CallStateFaxDetected
  - 18 - CallStateQueueFull
  - 19 - CallStateCleared
  - 20 - CallStateOverflowed
  - 21 - CallStateAbandoned
  - 22 - CallStateRedirected
  - 23 - CallStateForwarded
  - 24 - CallStateConsult
  - 25 - CallStatePickedup
  - 26 - CallStateDropped
  - 27 - CallStateDroppednoanswer

- 28 - CallStateUnknown
- 29 - CallStateCovered
- 30 - CallStateConverseOn
- 31 - CallStateBridged
- 32 - CallStateSilenceDetected
- 49 - CallStateDeafened
- 50 - CallStateHeld

Also, see genesys.ixn.callState ENUM Object.

Voice Events

The event namespace convention is voice.xxxx. The following are the Voice action result events:

Event	Attributes	Description
voice.createcall.done		This event indicates the success of the request and that the interaction has been created.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the interaction that was created.
	callstate	This is the Call Control Event Call State. (since 8.1.3, see below)
error.voice.createcall		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	interactionid	This is the ID associated with the interaction that might have been created in conjunction with this request, but has failed.
	error	This is the type of error that occurred: • timeout • invalidsource • invaliddestination • invalidserver • unknown
	description	This is a more detailed description of the error.
	callstate	This is the Call Control Event Call State.

Event	Attributes	Description
		(since 8.1.3, see below)
voice.hold.done		This event indicates the success of the request and that the specified resource's device is going on hold for the call.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the interaction that was created.
error.voice.hold		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	interactionid	This is the ID associated with the interaction that might have been created in conjunction with this request, but has failed.
	error	This is the type of error that occurred: • timeout • invalidsource • invalidinteraction • invalidserver • unknown
	description	This is a more detailed description of the error.
voice.retrieve.done		This event indicates the success of the request and that the specified resource's device is going to be retrieved from hold for the call.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with

Event	Attributes	Description
		the interaction that was created.
error.voice.retrieve		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	interactionid	This is the ID associated with the interaction that might have been created in conjunction with this request, but has failed.
	error	This is the type of error that occurred: • timeout • invalidsource • invalidinteraction • invalidserver • unknown
	description	This is a more detailed description of the error.
voice.consultation.done		This event indicates the success of the request and that the specified resource's device is going to put an existing call on hold and create a new call.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the interaction that was created.
error.voice.consultation		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.

Event	Attributes	Description
	interactionid	This is the ID associated with the interaction that might have been created in conjunction with this request, but has failed.
	error	This is the type of error that occurred: • timeout • invalidsource • invalidinteraction • invaliddestination • invalidserver • unknown
	description	This is a more detailed description of the error.
voice.alternate.done		This event indicates the success of the request and that the specified resource's devices are going to alternate between calls.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the interaction that was created.
error.voice.alternate		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	interactionid	This is the ID associated with the interaction that might have been created in conjunction with this request, but has failed.
	error	This is the type of error that occurred: • timeout • invalidsource

Event	Attributes	Description
		• invalidinteraction • invalidserver • unknown
	description	This is a more detailed description of the error.
voice.reconnect.done		This event indicates the success of the request and that the specified resource's devices are going to drop the active call and retrieve the held call.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction id associated with the interaction that was created.
error.voice.reconnect		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	interactionid	This is the ID associated with the interaction that might have been created in conjunction with this request, but has failed.
	error	This is the type of error that occurred: • timeout • invalidsource • invalidinteraction • invalidserver • unknown
	description	This is a more detailed description of the error.
voice.conference.done		This event indicates the success of the

Event	Attributes	Description
		request and that the specified resource's devices are going to conference two calls.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the interaction that was created.
error.voice.conference		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	interactionid	This is the ID associated with the interaction that might have been created in conjunction with this request, but has failed.
	error	This is the type of error that occurred: • timeout • invalidsource • invalidinteraction • invalidserver • unknown
	description	This is a more detailed description of the error.
voice.transfer.done		This event indicates the success of the request and that the specified resource's devices are going to transfer a call.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the interaction that was created.
error.voice.transfer		This indicates that an abnormal condition occurred while trying to perform the

Event	Attributes	Description
		request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	interactionid	This is the ID associated with the interaction that might have been created in conjunction with this request, but has failed.
	error	This is the type of error that occurred: • timeout • invalidsource • invalidinteraction • invalidserver • unknown
	description	This is a more detailed description of the error.
voice.privateservice.done		This event is sent when the request has been accepted by the underlying functional module and that it is able to send it. It is not an indication that the T-Server has handled or accepted the event.
	requestid	This is the ID associated with the request.
	interactionid	This is the ID associated with the interaction that was passed to the `<privateservice>` action.
error.voice.privateservice		This indicates that an abnormal condition occurred while trying to perform the request. This event may be seen as a result of failing to send the message or may occur after the target T-Server has received and attempted to process the event. In the case of failing to send the request it will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself as described in the error and the description. If this occurs as a result of an error in the processing of the request by

Event	Attributes	Description
		the T-Server the error property will be set to indicate "tserver", with a generic description and more details available within the properties.
	requestid	This is the ID associated with the request.
	interactionid	This is the ID associated with the interaction that was passed to the `<privateservice>` action.
	error	This is the type of error that occurred: • timeout • invalidsource • invalidinteraction • invalidserver • unknown tserver
	description	This is a more detailed description of the error.
	properties	This is an ECMAScript object that represents the various properties of the `TServer EventError` structure. It will only be present if the request has been successfully transmitted and received by the T-Server. If present, the error property will be set to "tserver" to signify this was an error generated by the T-Server handling the request.

callstate Attribute Considerations (since 8.1.3) The callstate property of the voice events is populated as follows:

- **callstate** - Call Control Event Call State - This is a detailed definition of the current call state of the interaction associated with `<createcall>` action events being raised when the type attribute value is "predictive". This property is an integer and may be undefined or have one of the following values:
  - 0 - CallStateOk
  - 1 - CallStateTransferred
  - 2 - CallStateConferenced
  - 3 - CallStateGeneralError
  - 4 - CallStateSystemError
  - 5 - CallStateRemoteRelease

- 6 - CallStateBusy
- 7 - CallStateNoAnswer
- 8 - CallStateSitDetected
- 9 - CallStateAnsweringMachineDetected
- 10 - CallStateAllTrunksBusy
- 11 - CallStateSitInvalidnum
- 12 - CallStateSitVacant
- 13 - CallStateSitIntercept
- 14 - CallStateSitUnknown
- 15 - CallStateSitNocircuit
- 16 - CallStateSitReorder
- 17 - CallStateFaxDetected
- 18 - CallStateQueueFull
- 19 - CallStateCleared
- 20 - CallStateOverflowed
- 21 - CallStateAbandoned
- 22 - CallStateRedirected
- 23 - CallStateForwarded
- 24 - CallStateConsult
- 25 - CallStatePickedup
- 26 - CallStateDropped
- 27 - CallStateDroppednoanswer
- 28 - CallStateUnknown
- 29 - CallStateCovered
- 30 - CallStateConverseOn
- 31 - CallStateBridged
- 32 - CallStateSilenceDetected
- 49 - CallStateDeafened
- 50 - CallStateHeld

Also, see genesys.ixn.callState ENUM Object.

Message Based Events

The event namespace convention is msgbased.xxxx. The following are the Msgbased action result events:

Event	Attributes	Description
msgbased.createmessage.done		This event indicates the success of the request and that a message and an associated interaction has been created.
	requestid	This is the ID associated with the request.
	interactionid	This is the interaction ID associated with the interaction that was created.
	srid	This is the ID of the suggested response that was used.
error.msgbased.createmessage		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred: • Timeout • Request Invalid
	description	This is a more detailed description of the error.
msgbased.sendmessage.done		This event indicates the success of the request and that a message and an associated interaction has been sent.
	requestid	This is the ID associated with the request.
error.msgbased.sendmessage		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a

Event	Attributes	Description
		result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred: • Timeout • Request Invalid
	description	This is a more detailed description of the error.

Chat Events

The event namespace convention is chat.xxxx. The following are the chat action result events:

Event	Attributes	Description
chat.sendchatmessage.done		This event indicates the success of the request and that a chat message has been sent in association with a given party.
	requestid	This is the ID associated with the request.
error.chat.sendchatmessage		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred: • Request Invalid
	description	This is a more detailed description of the error.
chat.gettranscript.done		This event indicates the success of the request and contains chat transcript.
	requestid	This is the ID associated with the request.
	transcript	This is the ECMAScript Object (array), which contains transcript messages as elements. Each element has the following properties: • date - number of seconds since 1 January 1970 00:00:00 UTC • device - name of chat party • text - chat message • visibility - specifies the

Event	Attributes	Description
		visibility level of this particular transcript event (could be: "ALL" – like conference mode, "INT" – like coaching mode, "VIP" – like monitoring mode for supervisors)
error.chat.gettranscript		This indicates that an abnormal condition occurred while trying to perform this request. This event will be sent as a result of a timeout of the request as well as due to problems with the request itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred.
	description	This is a more detailed description of the error.
chat.getcontent.done		This event indicates the success of the request and that the content object has been refreshed.
	requestid	This is the ID associated with the request.
error.chat.getcontent		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred: • Request Invalid
	description	This is a more detailed description of the error.

Dialog Interface

Overview

The dialog interface defines the functionality that allows the orchestration logic to do the following:

- Run a particular dialog application (VXML, HTML, and so on) on a specific interaction and by a specific resource/device.
- Collect the results of the dialog application that was run.

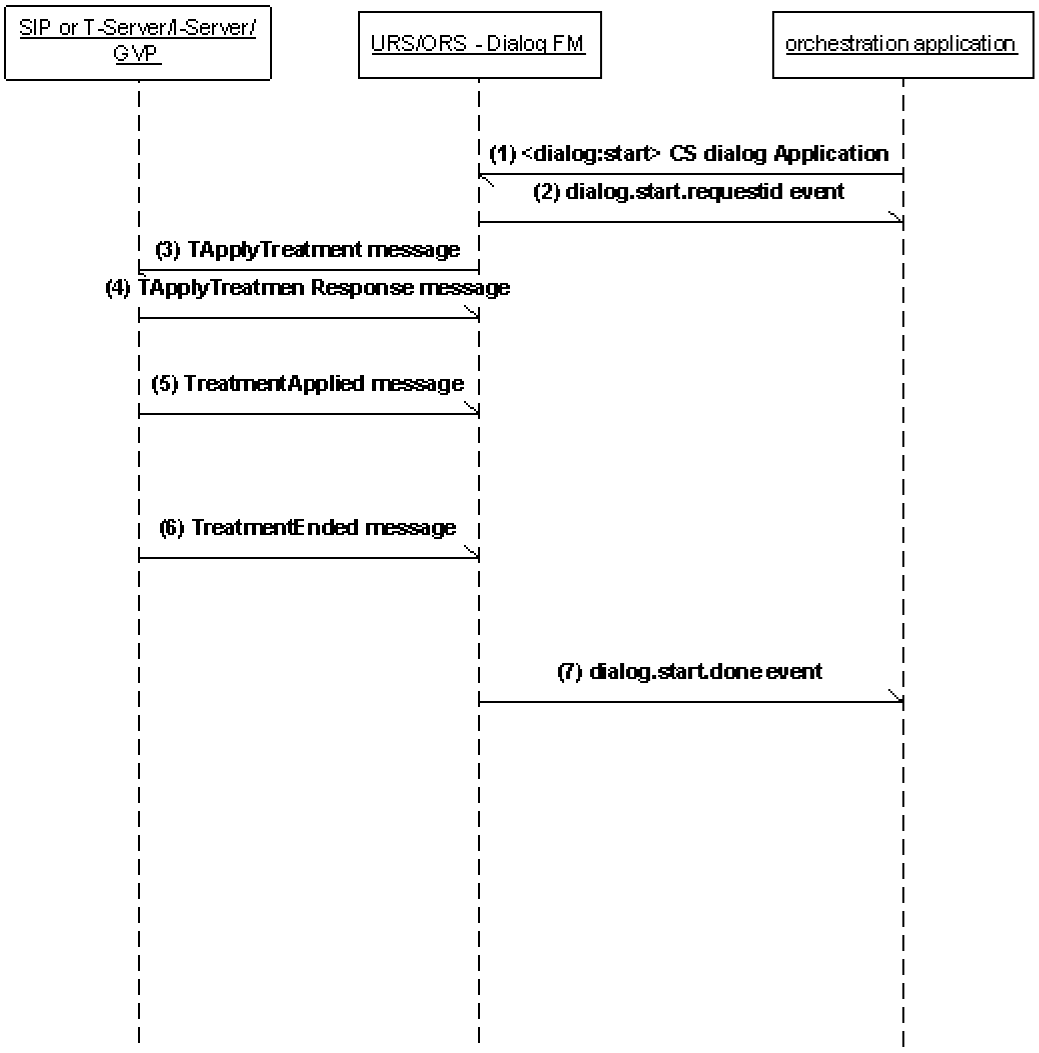
A dialog application represents the logic and presentation forms needed by the resources to have a media-specific conversation with a customer (for example, voice script, agent application (CRM, custom, and so on)). Dialog applications drive the behavior of resources in interacting with customers. The dialog application ***always*** operates on resources/devices. The actual presentation or communication (audio or visual) of the dialog application is media channel-specific. These dialog applications are actually communicated through the associated interactions and the resource's media device. These dialog applications drive the communication of the resources with the customer. This communication can be done directly by the resource itself, for example, by way of an IVR or a website. Or it can be done indirectly through a proxy, for example, the resource is the system's representation of the agent, the dialog application is the agent application, and the proxy is the actual agent. In this last case, the agent reads information from the agent application to the customer, collects information from the customer, and puts it into the agent application. The following are the types of dialog applications:

- Voice Script (VXML) - Directly - Voice interactions
- Agent Application (CRM, custom, agent script, and so on) - Indirectly - Can be used with any type of interaction, because it is indirectly communicated to the customer via a human business resource
- Web Application - Directly - Web interactions
- Legacy voice applications - Voice interactions
- Legacy voice treatments - Voice interactions

The granularity of a dialog application depends on how much interaction is needed with the orchestration logic layers. In some cases, the orchestration logic can actually be part of a dialog application.

Treatments are a specific type of dialog application. They are voice interaction related dialogs that are used to communicate with the customer while the orchestration logic is trying to find a resource or some other processing to help the customer (that is, while the customer is queued waiting). There will be a set of treatment-specific elements to cover this specific functionality. The treatment actions or elements can be used in combination with elements from the Queue functional module interface

The following is a sequence diagram to show the basic call flow the <start> action element which is generally the same for all the other treatments.


SIP or T-Server/I-Server/ GVP
URS/ORS - Dialog FM
orchestration application
(1) <dialog:start> CS dialog Application
(2) dialog.start.requestid event
(3) TApplyTreatment message
(4) TApplyTreatmen Response message
(5) TreatmentApplied message
(6) TreatmentEnded message
(7) dialog.start.done event

Dialog sequence diagram.png

Media Resource Addressing

The media resource device addresses that are used in in the dialog action elements must be known to the dialog functional module and must be configured in the Genesys configuration layer. These addresses vary based on the type of media by means of which they communicate with the dialog application. For example, voice interaction related dialog resource device addresses are directory or phone numbers. In an orchestration application, these resource device addresses will be represented in the following formats:

- String or Resource Object for Voice-related media dialog application with a format the same as the voice interaction FM Action element resource attributes. See Addressing Resources for details.

Extension Data

As of 8.1.2 when using the Dialog actions, extension data may be provided to influence the operation of the action and is passed using the hints attribute with a property named **extensions** containing the information that is required to be sent with the request. Also response events both error and normal done events may optionally carry extension data if returned from the device performing the operation. If extension data is available in such events it shall be found in the **extensions** property of the returned event.

Parameter Elements

We will have parameter elements for the following action elements to be used as input attributes.

- `<collect>`
- `<play>`
- `<playandcollect>`
- `<playandverify>`
- `<runtreatments>`

<prompts>

This is the top-level element which defines the set of prompts which are to be used under certain conditions associated with the treatment dialog.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
type	false	NMTOKEN	ann	tts, ann, iretry, ifailure, isuccess, itimeout, MSG_type	This specifies the type of condition that these prompts will be used for: • **ann** - This represents a list of normal announcements to be played to the caller. • **tts** - This represents a list of TTS-based announcements

Name	Required	Type	Default Value	Valid Values	Description
					to be played to the caller. All `<prompt>` element children must have the text attribute. • **iretry** - This represents a list of announcements to be played after input verification has failed and the caller is asked to reenter information. • **ifailure** - This represents a list of announcements to be played when the input verification has failed. • **isuccess** - This represents a list of announcements to be played when the input verification is successful. • **itimeout** - This represents

Name	Required	Type	Default Value	Valid Values	Description
					a list of announcements to be played when the time expires for collecting the input from the caller. • **MSG_type** - This represents the old MSG* user data key/value pairs that could be used for the `<play>` and `<playandcollect>` actions in place of the tts and ann prompt types. When the type attribute is set to this value, there can only be one child `<prompt>` element.

Children

- `<prompt>` - Occurs 1 to 10 times. Each instance defines a given announcement to be played to the caller.

<prompt>

This defines a given announcement to play to a caller.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
intid	false	value expression	none	Any expression that returns a valid integer	This represents a string ID for the announcement. This is mutually exclusive with the following attributes: • number • userAnnid • text See SCXML Legal Data Values and Value Expressions for details.
number	false	value expression	none	Any expression that returns a valid string	This represents the number to announce. The first digit defines how the number should be announced: • **0** - one at a time (example: 411 will be pronounced as four-one-one) • **1** - date (example: the eleventh of April) • **2** - time (four eleven a.m.) • **3** - phone number (four-one-one) • **4** - money (four dollars and eleven cents)

Name	Required	Type	Default Value	Valid Values	Description
					• **5** - number (four hundred and eleven) This is mutually exclusive with the following attributes: • intid • userAnnid • text See SCXML Legal Data Values and Value Expressions for details.
userannid	false	value expression	none	Any expression that returns a valid string	This represents the user announcement ID as returned after a successful recording of a user announcement request. This is mutually exclusive with the following attributes: • intid • number • text See SCXML Legal Data Values and Value Expressions for details.
userid	false	value expression	none	Any expression that returns a valid string	Used in conjuction with userannid to provide context in which userannid will be interpreted. This is often then the tenant name. (since 8.1.2)
text	false	value expression	none	Any expression that returns a valid	This represents the ASCII text to be

Name	Required	Type	Default Value	Valid Values	Description
				string	announced using text-to-speech technology (if supported by the IP equipment). This is mutually exclusive with the following attributes: • number • userAnnid • intid See SCXML Legal Data Values and Value Expressions for details.
interrupt	false	boolean expression	true	Any expression that returns a boolean (true, false)	This indicates whether the caller can interrupt the announcement. See SCXML Conditional Expressions for details.

Children

None

<input>

This is the top-level element which defines the characteristics needed to collect digits from a caller in association with the treatment dialog.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
max_digits	false	value expression	none	Any expression that returns a valid integer	The maximum number of digits to be collected (up to 31). **Note:** max_digits may be equal to 0. In this case, no time is spent waiting for caller input digits, and a response is returned indicating 0 digits were collected. See

Name	Required	Type	Default Value	Valid Values	Description
					SCXML Legal Data Values and Value Expressions for details.
abort_digits	false	value expression	none	Any expression that returns a valid string	This sequence of up to two keys aborts the digit-collection operation. If this sequence appears, the intelligent peripheral considers this to be a failed digit-collection attempt. See SCXML Legal Data Values and Value Expressions for details.
ignore_digits	false	value expression	none	Any expression that returns a valid string	This sequence of up to two keys is treated as though the keys have not been pressed. See SCXML Legal Data Values and Value Expressions for details.
backspace_digits	false	value expression	none	Any expression that returns a valid string	This sequence of up to two keys causes the previous keystroke to be discarded. See SCXML Legal Data Values and Value Expressions for details.
term_digits	false	value expression	none	Any expression that returns a valid string	This sequence of up to two keys causes all the digits, not including the term_digits, to be returned to the service logic as collected digits. See SCXML Legal Data Values and Value Expressions for details.
reset_digits	false	value expression	none	Any expression that returns a valid string	This sequence of up to two keys causes all the previous

Name	Required	Type	Default Value	Valid Values	Description
					keystrokes to be discarded. The digit collection resumes. See SCXML Legal Data Values and Value Expressions for details.
clear	false	boolean	false	true, false	Indicates whether any information that has been input should be cleared before digit collection starts. Not supported in GR-1129-CORE protocol implementation.
start_timeout	false	value expression	none	Any expression that returns a valid integer	The number of seconds the resource should wait for the caller to begin DTMF input. See SCXML Legal Data Values and Value Expressions for details.
digit_timeout	false	value expression	none	Any expression that returns a valid integer	The number of seconds the resource should wait between DTMF digits. See SCXML Legal Data Values and Value Expressions for details.
total_timeout	false	value expression	none	Any expression that returns a valid integer	The total number of seconds the resource should wait for the caller to provide the requested DTMF input See SCXML Legal Data Values and Value Expressions for details.

Children

None

<compare>

This defines what needs to be compared to the digits collected from the caller.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
digits	false	value expression	none	Any expression that returns a valid string	This represents the actual digits the caller input should be compared against. This is mutually exclusive with the following attributes: • userid • planid See SCXML Legal Data Values and Value Expressions for details.
userid	false	value expression	none	Any expression that returns a valid string	This represents the user ID string that the resource should use to index into a local table for verification. This is mutually exclusive with the following attributes: • digits • planid See SCXML Legal Data Values and Value Expressions for details.
planid	false	value expression	none	Any expression that returns a valid integer	This represents an index into a resource's table of dialing plans. The input is checked for general format compliance with the dialing plan selected by this ID. This is mutually exclusive with the following attributes: • digits

Name	Required	Type	Default Value	Valid Values	Description
					• userid See SCXML Legal Data Values and Value Expressions for details.
attempts	false	value expression	0	Any expression that returns a valid integer	This indicates the number of attempts the caller is allowed to make before failing the verification. See SCXML Legal Data Values and Value Expressions for details.
timeout	false	value expression	0	Any expression that returns a valid integer	The total number of seconds the resource should wait for the comparison to be made before timing out See SCXML Legal Data Values and Value Expressions for details.

Children

None

<pause>

This is the top-level element which defines the duration of a pause to be used between dialog treatments for the `<runtreatments>` element.

Attributes

Name	Required	Type	Default Value	Valid Values	Description
duration	false	value expression	0	Any expression that returns a valid integer	A value expression which returns the duration in seconds that the treatment processing should pause. See SCXML Legal Data Values and Value Expressions for

Name	Required	Type	Default Value	Valid Values	Description
					details.

Children

None

Action Elements

<runtreatments>

This action performs a series of treatments in the order specified. This action can only be a child of the Queue FM's `<submit>` action element.

Attribute Details

None

The following are action limitations:

- The `<stop>` action can not be used to cancel this series of treatments.
- This action can only be a child of the `<queue:submit>` action.
- No events will be generated for all treatment actions that are defined.

The following are examples:

```
<state id="Skills-Based-with-Treatments">
  <datamodel>
    <data id="reqid"/>
  </datamodel>
  <onentry>
    <queue:submit queue="'VQ1'" requestid="_data.reqid" ordertype="'min'"
                  orderstat="'StatExpectedWaitingTime'" timeout="100">
      <queue:targets type="agentgroup" statserver="'www.genesyslab.stserver1.com'">
        <queue:target name="'agtgrp2'"/>
      </queue:targets>
      <dialog:runtreatments>
        <dialog:play language="'English(US)'">
          <dialog:prompts type="ann">
            <dialog:prompt interrupt="true" intid="1111"/>
            <dialog:prompt interrupt="true" number="'2222'"/>
          </dialog:prompts>
        </dialog:play>
        <dialog:collect>
          <dialog:input max_digits="6" abort_digits="'1'" term_digits="'9'"
                        total_timeout="20"/>
        </dialog:collect>
        <dialog:playsound type="'music'" resource="'EMusicDN'" duration="100"/>
      </dialog:runtreatments>
    </queue:submit>
  </onentry>
  <transition event="queue.submit.done" target="next-state"/>
  <transition event="error.queue.submit" target="error-state"/>
</state>
```

Children

- **`<collect>`** - Occurs 0 to N. This action collects digits from a caller.
- **`<play>`** - Occurs 0 to N. This action plays announcements to a caller.
- **`<playandcollect>`** - Occurs 0 to N. This action plays announcements to a caller and then collects digits from a caller.
- **`<playandverify>`** - Occurs 0 to N. This action plays announcements to a caller, collects digits from the caller and then verifies that the digits are appropriate.
- **`<playsound>`** - Occurs 0 to N. This action plays sounds to a caller.
- **`<start>`** - Occurs 0 to N. This action executes a dialog application with which the caller will interact.
- **`<pause>`** - Occurs 0 to N. This action executes a pause in treatment processing. This is to be used in between the other treatments.
- **`<remote>`** - Occurs 0 to N. This action executes a dialog application with which the caller will interact by way of a new remote resource.
- **`<setdialogdefaultdest>`** - Occurs 0 to 1. This action informs the dialog application what the default destination of the interaction should be if a failure occurs between the orchestration platform and the dialog application resource.

Events

Events associated with the <runtreatments> action and the child action elements will not be generated.

<collect>

This action simply collects a set of digits from a caller. It is equivalent to the IRD function block "Collect Digits".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as

Name	Required	Type	Default Value	Valid Values	Description
					part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.collect.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID from which the digits will be collected. There is a special value that can be returned: • **"0"** means the functional module will use the `_genesys.FMname.interac` for collecting the digits.

Name	Required	Type	Default Value	Valid Values	Description
					See SCXML Legal Data Values and Value Expressions for details.
device	false	value expression	none	Any expression that returns a valid string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- The `<stop>` action can not be used to cancel this treatment.

- If this action element is a child of the `<runtreatments>` element then the following attributes are not evaluated and should not be used. The similar attributes in the associated Queue FM's `<submit>` and`<update>` action elements will be used instead.
  - requestid
  - interactionid

The following is an example:

```
<state id="Collect-Digits">
  <onentry>
    <dialog:collect device="'dn1001'">
      <dialog:input max_digits="6" abort_digits="'1'" term_digits="'9'"
                    total_timeout="20"/>
    </dialog:collect>
  </onentry>
  <transition event="dialog.collect.done" target="next-state"/>
  <transition event="error.dialog.collect" target="error-state"/>
</state>
```

Children

- **`<input>`** - Occurs 1 time. This instance defines the characteristics needed to collect the digits.

Events

The following events can be generated as part of this action:

- `dialog.collect.done` - This event will be sent when the digits have been collected.
- `dialog.collect.requestid` - This event will be sent when the collect treatment has started.
- `error.dialog.collect` - This event will be sent as a result of problems with the request itself.

<play>

This action simply plays a set of announcements to a caller. It is equivalent to the IRD function block "Play Announcement" and "Text to Speech".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The

Name	Required	Type	Default Value	Valid Values	Description
					location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.play.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.
Interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID on which the announcements will be played. There is a special value that can be returned: <ul><li>**"0"** means the functional module will use `_genesys.FMname.interac` for collecting the digits.</li></ul> See SCXML Legal Data Values and Value Expressions for details.
language	false	value expression	English(US)	Any expression that returns a	A value expression which returns a

Name	Required	Type	Default Value	Valid Values	Description
				string with one of the following values: English (US), Spanish, Mandarin, Cantonese, Vietnamese, French, French (Canada), German, Italian, Japanese, Korean, Russian	string specifying a language in which the announcements should be made. See SCXML Legal Data Values and Value Expressions for details.
device	false	value expression	none	Any expression that returns a valid string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- The `<stop>` action can not be used to cancel this treatment.
- If this action element is a child of the `<runtreatments>` element then the following attributes are not evaluated and should not be used. The similar attributes in the associated Queue FM's `<submit>` and `<update>` action elements will be used instead.
    - requestid
    - interactionid

The following is an example:

```
<state id="Play-Message">
  <onentry>
    <dialog:play language="'English(US)'">
      <dialog:prompts type="ann">
        <dialog:prompt interrupt="true" intid="1111"/>
        <dialog:prompt interrupt="true" number="'2222'"/>
      </dialog:prompts>
    </dialog:play>
  </onentry>
  <transition event="dialog.play.done" target="next-state"/>
  <transition event="error.dialog.play" target="error-state"/>
</state>
```

Children

- **`<prompts>`** - Occurs 1 time. This instance defines the set of announcements to be played to the caller. The `<prompts>`'s type attribute can only be set to "ann" or "tts"

Events

The following events can be generated as part of this action:

- `dialog.play.done` - This event will be sent when the message has been played.
- `dialog.play.requestid` - This event will be sent when the play treatment has started.
- `error.dialog.play` - This event will be sent as a result of problems with the request itself.

<playandcollect>

This action plays a set of announcements to a caller and then collects digits. In addition, you can specify the appropriate attributes to verify the digits collected against a defined set of digits. It is equivalent to the IRD function blocks "Play Announcement and Collect Digits" and "Text to Speech and Collect Digits".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location	This is the location

Name	Required	Type	Default Value	Valid Values	Description
				expression	for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.playandcollect.requestid event is received.If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID on which the announcements will be played and the digits collected. There is a special value that can be returned: • **"0"** means the functional module will use `_genesys.FMname.interac`

Name	Required	Type	Default Value	Valid Values	Description
					for collecting the digits. See SCXML Legal Data Values and Value Expressions for details.
language	false	value expression	English (US)	Any expression that returns a string with one of the following values: English (US), Spanish, Mandarin, Cantonese, Vietnamese, French, French (Canada), German, Italian, Japanese, Korean, Russian	A value expression which returns a string specifying a language in which the announcements should be made. See SCXML Legal Data Values and Value Expressions for details.
device	false	value expression	none	Any expression that returns string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to

Name	Required	Type	Default Value	Valid Values	Description
					the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- The `<stop>` action can not be used to cancel this treatment.
- If this action element is a child of the `<runtreatments>` element, then the following attributes are not evaluated and should not be used. The similar attributes in the associated Queue FM's `<submit>` and`<update>` action elements will be used instead.
  - requestid
  - Interactionid

The following is an example:

```
<state id="Play-and-Collect">
  <onentry>
    <dialog:playandcollect language="'English(US)'">
      <dialog:prompts type="ann">
        <dialog:prompt interrupt="true" intid="1111"/>
        <dialog:prompt interrupt="true" number="'2222'"/>
      </dialog:prompts>
      <dialog:input max_digits="4" abort_digits="'1'" term_digits="'9'"
                    total_timeout="20"/>
    </dialog:playandcollect>
  </onentry>
  <transition event="dialog.playandcollect.done" target="next-state"/>
  <transition event="error.dialog.playandcollect" target="error-state"/>
</state>
```

Children

- **`<prompts>`** - Occurs 1 time. This instance defines the set of announcements to be played to the caller. The `<prompts>`'s type attribute can only be set to "ann" or "tts".
- **`<input>`** - Occurs 1 time. This instance defines the characteristics needed to collect the digits.

Events

The following events can be generated as part of this action:

- `dialog.playandcollect.done` - This event will be sent when the messages have been played and the associated digits have been collected.

- `dialog.playandcollect.requestid` - This event will be sent when the playandcollect treatment has started.
- `error.dialog.playandcollect` - This event will be sent as a result of problems with the request itself.

<playandverify>

This action plays a set of announcements to a caller, collects digits, and verifies the digits against a defined set of digits. It is equivalent to the IRD function block "Verify Digits".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.playandverify.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.

Name	Required	Type	Default Value	Valid Values	Description
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID on which the announcements will be played and the digits will be collected and verified. There is a special value that can be returned: • **"0"** means the functional module will use `_genesys.FMname.interac` for collecting the digits. See SCXML Legal Data Values and Value Expressions for details.
language	false	value expression	English (US)	Any expression that returns a string with one of the following values: English (US), Spanish, Mandarin, Cantonese, Vietnamese, French, French (Canada), German, Italian, Japanese, Korean, Russian	A value expression which returns a string specifying a language in which the announcements should be made. See SCXML Legal Data Values and Value Expressions for details.
device	false	value expression	none	Any expression that returns string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)

Name	Required	Type	Default Value	Valid Values	Description
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- The `<stop>` action can not be used to cancel this treatment.
- With this action you cannot use a `<prompts>` type attribute of "tts".
- If this action element is a child of the `<runtreatments>` element, then the following attributes are not evaluated and should not be used. The similar attributes in the associated Queue FM's `<submit>` and `<update>` action elements will be used instead.
  - requestid
  - interactionid

The following is an example:

```
<state id="Play-and-Verify">
  <onentry>
    <dialog:playandverify language="'English(US)'">
      <dialog:prompts type="ann">
        <dialog:prompt interrupt="true" intid="1111"/>
        <dialog:prompt interrupt="true" number="'2222'"/>
      </dialog:prompts>
      <dialog:prompts type="iretry">
        <dialog:prompt interrupt="true" intid ="3333"/>
      </dialog:prompts>
      <dialog:prompts type="isuccess">
        <dialog:prompt intid="4444"/>
        <dialog:prompt interrupt="true" text="'hi'"/>
      </dialog:prompts>
      <dialog:prompts type="ifailure">
        <dialog:prompt interrupt="true" intid="1111"/>
        <dialog:prompt interrupt="true" userannid="10"/>
      </dialog:prompts>
      <dialog:prompts type="itimeout">
        <dialog:prompt text="'timeout'"/>
      </dialog:prompts>
      <dialog:input max_digits="10" abort_digits="'1'" term_digits="'9'"
                    total_timeout="20"/>
      <dialog:compare digits="'12426'" attempts="3"/>
    </dialog:playandverify>
  </onentry>
  <transition event="dialog.playandverify.done" target="next-state"/>
  <transition event="error.dialog.playandverify" target="error-state"/>
</state>
```

Children

- **`<prompts>`** - Must occur at least once and up to 5 times in total, each instance must have a unique type attribute of either 'ann', 'iretry', 'isuccess', 'ifailure' or 'itimeout', where a prompt type attribute of 'ann' being declared to be mandatory and **MUST** be provided in all `<dialog:playandverify>`. All other prompt types remain **optional**. You may only have a maximum of one type attribute within the series of `<dialog:prompts>` within `<dialog:playandverify>` given that they are required to be unique, defining multiple "iretry" types for example or "ann" within the same `<dialog:playandverify>` is not supported and the behavior is undetermined. These instances define the different sets of announcements to be played to the caller based on the verification process.
- **`<input>`** - Occurs 1 time. This instance defines the characteristics needed to collect the digits.
- **`<compare>`** - Occurs 1 time. This instance defines what needs to be compared to the digits collected from the caller.

Events

The following events can be generated as part of this action:

- `dialog.playandverify.done` - This event will be sent when the messages are played, the digits have been collected and verified.
- `dialog.playandverify.requestid` - This event will be sent when the playandverfy treatment has started.
- `error.dialog.playandverify` - This event will be sent as a result of problems with the request itself.

<playsound>

This action plays a given voice-related sound to a caller. It is a voice-specific treatment. It is equivalent to the following IRD function blocks

- "Busy"
- "Fast Busy"
- "Music"

- "Ringback"
- "Silence"
- "RAN"

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.playsound.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID on which to play the sound. There is a special value that can be returned:

Name	Required	Type	Default Value	Valid Values	Description
					• **"0"** means the functional module will use _genesys.FMname.interact for collecting the digits. See SCXML Legal Data Values and Value Expressions for details.
type	true	value expression	none	busy, fastbusy, music, ringback, silence, ran	A value expression which returns a string specifying the type of sound to play to the caller. See SCXML Legal Data Values and Value Expressions for details.
resource[2]	false	value expression	none	Any valid string or Resource Object	For type equal to "music", this will be the source of the music (that is, MUSIC_DN), and is a required attribute unless SIP Server has been configured with a default file[1]. To specify the number of repetitions, the parameter `repeat`=<N> must be used, where <N> is any positive integer. If no repetition is specified, the music file loops forever. The valid formats are: • `<directory>/<music file name>` - The specified file loops endlessly. • `<directory>/<music file`

Name	Required	Type	Default Value	Valid Values	Description
					`name>;repeat=<N>` -The specified file is repeated <N> times. [1] The `TServer/default-music` option in the SIP Server configuration object is used if the `resource` is not specified. For type equal to "ran", this will be the source of the recorded announcement (that is, ROUTE) and is a required attribute. See SCXML Legal Data Values and Value Expressions for details.
duration[2]	false	value expression	0	Any expression that returns a valid integer	A value expression which returns the duration in seconds that the sound should be played. See SCXML Legal Data Values and Value Expressions for details.
device	false	value expression	none	Any expression that returns string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object

Name	Required	Type	Default Value	Valid Values	Description
					containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

[2] For playing music, if both repeat (in resource) and duration are specified, repeat takes priority.
Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- The `<stop>` action can not be used to cancel this treatment.
- The following types of sounds are hard coded to be not interruptible:
  - busy, fastbusy, ran
- The following types of sounds are hard coded to be interruptible:
  - music, ringback, silence
- If this action element is a child of the `<runtreatments>` element, then the following attributes are not evaluated and should not be used. The similar attributes in the associated Queue FM's `<submit>` and `<update>` action elements will be used instead.
  - requestid
  - Interactionid

The following are some examples:

```
<state id="Play-Sound-Busy">
  <onentry>
    <dialog:playsound type="'busy'" duration="100" device="'dn1001'"/>
  </onentry>
  <transition event="dialog.playsound.done" target="next-state"/>
  <transition event="error.dialog.playsound" target="error-state"/>
```

```
</state>
<state id="Play-Sound-Music-30sec">
  <onentry>
    <dialog:playsound type="'music'" resource="'path_to/ElevatorMusicDN'" duration="30"/>
  </onentry>
  <transition event="dialog.playsound.done" target="next-state"/>
  <transition event="error.dialog.playsound" target="error-state"/>
</state>
<state id="Play-Sound-Music-Once">
  <onentry>
    <dialog:playsound type="'music'" resource="'path_to/ElevatorMusicDN;repeat=1'"/>
  </onentry>
  <transition event="dialog.playsound.done" target="next-state"/>
  <transition event="error.dialog.playsound" target="error-state"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `dialog.playsound.done` - This event will be sent when the sound has been played.
- `dialog.playsound.requestid` - This event will be sent when the play sound treatment has started.
- `error.dialog.playsound` - This event will be sent as a result of problems with the request itself.
- `error.dialog.playsound.timeout` - This event is sent by URS after the number of seconds specified in the 'duration' attribute has elapsed, and URS has not received the treatment.done event for playing the sound from Tserver.

<createann>

This action creates and records an announcement from a user. It is equivalent to the IRD function block "Record User Announcement". This is primarily used to collect a message from a customer.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an

Name	Required	Type	Default Value	Valid Values	Description
					internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.createann.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID on which to create or record the announcement. There is a special value that can be returned: • **"0"** means the functional module will use `_genesys.FMname.interac` for collecting the digits. See SCXML Legal Data Values and Value Expressions for details.
userid	true	value expression	none	Any expression that returns a valid string	A value expression which returns a string specifying the user ID

Name	Required	Type	Default Value	Valid Values	Description
					associated with this recording announcement. See SCXML Legal Data Values and Value Expressions for details.
abort_digits	false	value expression	none	Any expression that returns a valid string	A value expression which returns the sequence of up to two keys that the caller can enter to abort the recording process. The IP is to consider this as a failed recording attempt. See SCXML Legal Data Values and Value Expressions for details.
term_digits	false	value expression	none	Any expression that returns a valid string	A value expression which returns the sequence of up to two keys that the caller can enter to indicate that the caller has finished recording the announcement. See SCXML Legal Data Values and Value Expressions for details.
reset_digits	false	value expression	none	Any expression that returns a valid string	A value expression which returns the sequence of up to two keys that the caller can enter to restart the recording announcement. Any announcement recorded up to the point of these keystrokes will be discarded. See SCXML Legal Data Values and Value Expressions for details.
start_timeout	false	value expression	none	Any expression that returns a valid integer	A value expression which returns the number of seconds the resource should wait for the

Name	Required	Type	Default Value	Valid Values	Description
					callers to begin recording their announcements. See SCXML Legal Data Values and Value Expressions for details.
total_timeout	false	value expression	300	Any expression that returns a valid integer	A value expression which returns the total number of seconds the resource should wait for the callers to finish recording their announcements. (since 8.1.200.43, ORS will set a default value of 300 seconds for total_timeout) See SCXML Legal Data Values and Value Expressions for details.
device	false	value expression	none	Any expression that returns string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so

Name	Required	Type	Default Value	Valid Values	Description
					on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- The `<stop>` action can not be used to cancel this treatment.

The following is an example:

```
<state id="Create-Announcement">
  <onentry>
    <dialog:createann userid="'12334567'" abort_digits="'1'" term_digits="'9'"
                      total_timeout="20" device="'dn1001'">
      <dialog:prompts type="ann">
        <dialog:prompt interrupt="true" intid="1111"/>
        <dialog:prompt interrupt="true" number="'2222'"/>
      </dialog:prompts>
    </dialog:createann>
  </onentry>
  <transition event="dialog.createann.done" target="next-state"/>
  <transition event="error.dialog.createann" target="error-state"/>
</state>
```

Children

- `<prompts>` - Occurs 1 time. This instance defines the set of announcements to be played to the caller. The `<prompts>`'s type attribute can only be set to "ann".

Events

The following events can be generated as part of this action:

- `dialog.createann.done` - This event will be sent when the announcement has been created.
- `dialog.createann.requestid` - This event will be sent when the create announcement treatment has started.
- `error.dialog.createann` - This event will be sent as a result of problems with the request itself.

<deleteann>

This action creates and records an announcement from a user. It is equivalent to the IRD function block "Delete User Announcement"

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestlid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.deleteann.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID on which to create or record the announcement. There is a special value that can be returned:

Name	Required	Type	Default Value	Valid Values	Description
					• **"0"** means the functional module will use `_genesys.FMname.interac` for collecting the digits. See SCXML Legal Data Values and Value Expressions for details.
userid	true	value expression	none	Any expression that returns a valid string	A value expression which returns a string specifying the user ID associated with the announcement to be deleted. See SCXML Legal Data Values and Value Expressions for details.
annid	true	value expression	none	Any expression that returns a valid integer	A value expression which returns the user announcement ID as returned in the dialog.createann.done event. See SCXML Legal Data Values and Value Expressions for details.
device	false	value expression	none	Any expression that returns string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)
hints	false	value expression	none	Any valid	A value expression

Name	Required	Type	Default Value	Valid Values	Description
				ECMAScript object	which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- The `<stop>` action can not be used to cancel this treatment.
- The action is hard coded to be not interruptible.

The following is an example:

```
<state id="Delete-Announcement">
  <onentry>
    <dialog:deleteann userid="'12334567'" annid="464646464"/>
  </onentry>
  <transition event="dialog.deleteann.done" target="next-state"/>
  <transition event="error.dialog.deleteann" target="error-state"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `dialog.deleteann.done` - This event will be sent when the announcement have been deleted.
- `dialog.deleteann.requestid` - This event will be sent when the delete announcement treatment has started.
- `error.dialog.deleteann` - This event will be sent as a result of problems with the request itself.

<start>

This action requests that a specific dialog be started on a specific interaction and by a specific resource. This is used for all different types of media. This action is equivalent to the IRD function block "Play Application".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.start.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.

Name	Required	Type	Default Value	Valid Values	Description
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID for which the dialog is to be started. There is a special value that can be returned: • **"0"** means the functional module will use _genesys.FMname.interact to route the interaction. See SCXML Legal Data Values and Value Expressions for details.
type	true	value expression	none	vxml, applid	A value expression which returns the type of application that is to be started. The following are the different types that will be supported: • **vxml** - This is a VXML document that is to be started on the voice platform resource. • **applid** - This indicates that the request is for the "Play Application" IRD function block. See SCXML Legal Data Values and Value Expressions for details.

Name	Required	Type	Default Value	Valid Values	Description
application	false	value expression		Any expression that returns a valid integer (for applid) or string (for vxml)	A value expression which returns the identifier of a new application which is to be started by the resource in association with the interaction. The format of the application reference is specific to each dialog type: • **vxml** - This is the URL for the VXML document. • **applid** - This is the APP_ID parameter from the "PlayApplication" IRD function block. If this attribute is not specified, then the functional module will use the associated `<param>` elements. See SCXML Legal Data Values and Value Expressions for details.
device	false	value expression	none	Any expression that returns string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)
hints	false	value expression	none	Any valid	A value expression

Name	Required	Type	Default Value	Valid Values	Description
				ECMAScript object	which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- This action can be cancelled using the `<stop>` action.
- For dialog type of "applid", the timeout attribute will be ignored.
- For dialog type of "applid" the parameters supplied in the name list can only be of simple types.
- All `<param>` name attribute values must match the case of the parameter name of the underlying dialog system (for all Genesys treatments, parameters are in upper case - "LANGUAGE").
- The following is the data model for the dialog type of "appid" (that is, the "Play Application" function block action)
  - `<param name="LANGUAGE"/>` - The "Play Application" function block's LANGUAGE parameter
  - `<param name="parameterN"/>` - Any custom parameter of the "Play Application" function block - You can add any parameter you want, but this function block functionality only supports strings and integer parameters.

Note: the application attribute will be used for the "Play Application" function block's APP_ID parameter.

- If this action element is a child of the `<runtreatments>` element, then the following attributes are not evaluated and should not be used. The similar attributes in the associated Queue FM's `<submit>` and `<update>` action elements will be used instead.

- requestid
- Interactionid

Children

- `<param>` Occurs 0 to N. See SCXML `<param>` for details. These parameters will be submitted to the processing resource, based on the functional module's underlying protocol (for example, it will be via key/value pairs for T-Server).

Events

The following events can be generated as part of this action:

- `dialog.start.done` - This event will be sent when the started dialog application is complete.
- `dialog.start.requestid` - This event will be sent when the dialog application has started.
- `error.dialog.start` - This event will be sent as a result of a timeout of the request with an error attribute value of timeout, as well as due to problems with the request itself.

Examples

The following is an example of using the IRD Play Application function block:

```
<state id="Play-Application">
  <onentry>
    <dialog:start type="'applid'" application="464646464">
      <param name="APP_URL" expr="'www.bigplanes.com\bestinworld'"/>
    </dialog:start>
  </onentry>
  <transition event="dialog.start.done" target="next-state"/>
  <transition event="error.dialog.start" target="error-state"/>
</state>
```

The following is an example of using VoiceXML as a treatment:

```
<state id="Play-Application">
  <onentry>
    <dialog:start type="'vxml'" >
      <param name="APP_URI" expr="'http://gvphost/treatment.vxml'"/>
    </dialog:start>
  </onentry>
  <transition event="dialog.start.done" target="next-state"/>
  <transition event="error.dialog.start" target="error-state"/>
</state>
```

<stop>

This action will terminate call, referenced by interactionid attribute. However, it is supported by very few TServers, so it is better to use "terminate" action from "interaction" extension for that purpose. This action is equivalent to the IRD function block "Cancel Call".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	true	value expression	none	Any valid location expression	A value expression which returns the ID of the previously started dialog-related action. Legal Data Values and Value Expressions for details.
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID for which the dialog is to be started. There is a special value that can be returned: • **"0"** means the functional module will use _genesys.FMname.interact to route the interaction. This attribute is only valid if the compatible attribute is true. See SCXML Legal Data Values and Value Expressions for details.
compatible	false	boolean	false	true, false	This value indicates whether the action is compatible with older capabilities. If **true**, then this will do the "IRD Cancel Call" function. If **false**, then this will just stop or terminate the dialog and the actual interaction.
device	false	value expression	none	Any expression that returns string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to

Name	Required	Type	Default Value	Valid Values	Description
					URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- Only supports a compatible attribute value of true.

The following is an example for IRD "Cancel Call":

```
<state id="Stop-Application">
  <datamodel>
    <data id="reqid"/>
  </datamodel>
  <onentry>
    <dialog:stop requestid="_data.reqid" compatible="true"/>
  </onentry>
  <transition event="dialog.stop.done" target="next-state"/>
  <transition event="error.dialog.stop" target="error-state"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `dialog.stop.done` - This event will be sent when the dialog application stop action has completed.
- `error.dialog.stop` - This event will be sent as a result of problems with the request itself.
- `error.dialog.start` - This event is sent for the dialog action that was started. **Note:** The error.dialog.start events will be sent before the queue.stop.done.

<remote>

This action requests that a specific dialog be started on a specific interaction and by a new remote resource. This is used for voice media only. This action is equivalent to the IRD function block "IVR".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.remote.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the

Name	Required	Type	Default Value	Valid Values	Description
					completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every request must receive a unique identifier.
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression that returns the interaction ID for which the dialog is to be started. There is a special value that can be returned: • **"0"** means the functional module will use _genesys.FMname.interact to route the interaction. See SCXML Legal Data Values and Value Expressions for details.
application	false	value expression	none	Any expression that returns a valid string	A value expression which returns the identifier of a new application that is to be started by the new resource in association with the interaction. See SCXML Legal Data Values and Value Expressions for details.
duration	false	value expression	0	A value expression which returns an integer	A value expression which returns an integer that represents the number of seconds to wait. See SCXML Legal Data Values

Name	Required	Type	Default Value	Valid Values	Description
					and Value Expressions for details. The character string returned must be interpreted as a time interval. This interval begins when `<remote>` is executed. A failed and timed out submit must return the error.remote.start event.
destination	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the address name of a new remote resource which will be added to provide the treatment on the interaction. See SCXML Legal Data Values and Value Expressions for details.
default	false	value expression		Any value expression that returns a valid string	A value expression which returns the destination that should be used if there are problems transferring the interaction to the destination for treatment. See SCXML Legal Data Values and Value Expressions for details.
compatible	false	boolean	false	false, true	This value indicates whether the action is compatible with an attribute signature. If **true**, the following are the valid attributes: • application - SCRIPT • destination - TARGET • timeout -

Name	Required	Type	Default Value	Valid Values	Description
					DURATION If **false**, the following are the valid attributes: • destination - LABEL • default - DNIS
device	false	value expression	none	Any expression that returns string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. Will be ignored if compatible is set to true. (since 8.1.2)
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. Applicable only if compatible is false. (Since 8.1.2)

The following are action limitations:

- The `<stop>` action can not be used to cancel this treatment.
- If the compatible attribute is **true**, the following are the valid attributes:
  - application - SCRIPT
  - destination - TARGET
  - duration - DURATION
- If the compatible attribute is **false**, the following are the valid attributes:
  - destination - LABEL
  - default - DNIS
- If this action element is a child of the `<runtreatments>` element then the following attributes are not evaluated and should not be used. The similar attributes in the associated Queue FM's `<submit>` and`<update>` action elements will be used instead.
  - requestid
  - interactionid

The following is an example for compatible = false:

```
<state id="Remote-compatible-false">
  <onentry>
    <dialog:remote destination="'123456'" default="'2334'"/>
  </onentry>
  <transition event="dialog.remote.done" target="next-state"/>
  <transition event="error.dialog.remote" target="error-state"/>
</state>
```

The following is an example for compatible = true:

```
<state id="Remote-compatible-true">
  <onentry>
    <dialog:remote compatible="true"
                   destination="'2323@www.genesys.com\server1.AG'"
                   application="'mortgage'"/>
  </onentry>
  <transition event="dialog.remote.done" target="next-state"/>
  <transition event="error.dialog.remote" target="error-state"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `dialog.remote.done` - This event will be sent when the remote dialog application is complete.
- `dialog.remote.requestid` - This event will be sent when the remote dialog treatment has started.
- `error.dialog.remote` - This event will be sent as a result of a timeout of the request with an error attribute value of timeout, as well as due to problems with the request itself.

<setdialogdefaultdest>

This action requests a Genesys-specific treatment action. This action is equivalent to the IRD function block ""Set Default Destination".

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. This value will only be valid when the dialog.setdialogdefaultdest.requestid event is received. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the action completion event. Every

Name	Required	Type	Default Value	Valid Values	Description
					request must receive a unique identifier.
interactionid	false	value expression	"0"	Any expression that returns a valid string	A value expression which returns the interaction ID for which the action is to be done. There is a special value that can be returned: • **"0"** means the functional module will use _genesys.FMname.interact to route the interaction. See SCXML Legal Data Values and Value Expressions for details.
destination	true	value expression	none	Any value expression that returns a valid string	A value expression which returns the address name of the new remote resource that will be added to provide the treatment on the interaction. See SCXML Legal Data Values and Value Expressions for details.
device	false	value expression	none	Any expression that returns string or object	If specified ORS will play treatments itself, otherwise, treatment playing is delegated to URS. The device should specify the DN where the call is currently located. If the call is on multiple DNs, specify the DN for which the treatment will be applied. (since 8.1.2)

Name	Required	Type	Default Value	Valid Values	Description
hints	false	value expression	none	Any valid ECMAScript object	A value expression which returns the ECMAScript object containing information which may be used by the implementing functional module when applying dialog action to interaction. This information may consist of protocol-specific parameters, protocol selection guidelines, and so on. **Note:** The meaning of these hints is specific to the implementing functional module. See SCXML Legal Data Values and Value Expressions for details. (since 8.1.2)

Attribute **hints** considerations

- Property **extensions** of **hints** object allow to specify content of AttributeExtension of resulted RequestRouteCall. Valid value for property **extensions** is ECMAScript object. (Since 8.1.2)

The following are action limitations:

- The `<setdialogdefaultdest>` action can not be used to cancel this treatment.
- If this action element is a child of the `<runtreatments>` element then the following attributes are not evaluated and should not be used. The similar attributes in the associated Queue FM's `<submit>` and`<update>` action elements will be used instead.
  - requestid
  - interactionid

The following is an example:

```
<state id="Set-Default-Destination">
  <onentry>
    <dialog:setdialogdefaultdest destination="'12345'"/>
  </onentry>
  <transition event="dialog.setdialogdefaultdest.done" target="next-state"/>
  <transition event="error.dialog.setdialogdefaultdest" target="error-state"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `dialog.setdialogdefaultdest.done` - This event will be sent when the default destination is set.
- `dialog.setdialogdefaultdest.requestid` - This event will be sent when the set default destination treatment has started.
- `error.dialog.setdialogdefaultdest` - This event will be sent as a result of a timeout of the request with an error attribute value of timeout, as well as due to problems with the request itself.

Events

The following are the dialog action result events.

Event	Attributes	Description
dialog.start.done		This event indicates the successful completion of the starting of an application. This can be sent in conjunction with the `<start>` action or asynchronously to indicate the completion of an application started at the resource.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.start.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.start		This indicates that an abnormal condition occurred while trying to perform the start request. This event will be sent as a result of a timeout of the request as well as problems with the request itself.
	requestid	This is the ID associated with the request.

Event	Attributes	Description
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying to process this dialog action. • cancelled - This dialog action has been cancelled. • timeout - A target has not been found within the requested time period xxxx. xxxx is the value of the timeout attribute.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.stop.done		This event indicates the success of the stop request and that the request has stopped the dialog application.
	requestid	This is the ID of the <`stop`> request.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)

Event	Attributes	Description
error.dialog.stop		This indicates that an error occurred while trying to perform the `<stop>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • invalidrequestid - The request ID xxxx does not match any outstanding dialog action requests. xxxx is the value of the requestid attribute.
	description	This is a more detailed description of the error.
	requestid	This is the ID of the `<continue>` request.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.collect.done		This event indicates the success of the collect request and collects the digits.
	requestid	This is the ID of the `<collect>` request.
	digits	These are the digits that were collected.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.collect.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.collect		This indicates that an error occurred while trying to perform the `<collect>` request.

Event	Attributes	Description
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying to process this dialog action. • cancelled - This dialog action has been cancelled.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.play.done		This event indicates the success of the play request and that the announcements were played.
	requestid	This is the ID of the `<play>` request.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.play.requestid		This event provides the application with request ID for the given request that was

Event	Attributes	Description
		invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.play		This indicates that an error occurred while trying to perform the `<play>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying to process this dialog action. • cancelled - This dialog action has been cancelled.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.playandcollect.done		This event indicates the success of the playandcollect request and that the announcements were played and the

Event	Attributes	Description
		digits collected.
	requestid	This is the ID of the `<playandcollect>` request.
	digits	These are the digits that were collected.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.playandcollect.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.playandcollect		This indicates that an error occurred while trying to perform the `<playandcollect>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying

Event	Attributes	Description
		to process this dialog action. • cancelled - This dialog action has been cancelled.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.playandverify.done		This event indicates the success of the playandverify request and that the announcements were played and the digits were collected and verified.
	requestid	This is the ID of the `<playandverify>` request.
	digits	These are the digits that were collected.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.playandverify.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.playandverify		This indicates that an error occurred while trying to perform the `<playandverify>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request.

Event	Attributes	Description
		yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying to process this dialog action. • cancelled - This dialog action has been cancelled.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.playsound.done		This event indicates the success of the playsound request and that the sound was played.
	requestid	This is the ID of the `<playsound>` request.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.playsound.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.playsound		This indicates that an error occurred while trying to perform the `<playsound>` request.

Event	Attributes	Description
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying to process this dialog action. • cancelled - This dialog action has been cancelled.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.createann.done		This event indicates the success of the createann request and that the announcement for the user was recorded.
	requestid	This is the ID of the `<createann>` request.
	userannid	This is the ID of the announcement that was recorded for the user.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)

Event	Attributes	Description
dialog.createann.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.createann		This indicates that an error occurred while trying to perform the `<createann>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying to process this dialog action. • cancelled - This dialog action has been cancelled.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)

Event	Attributes	Description
dialog.deleteann.done		This event indicates the success of the deleteann request and that the announcement was deleted.
	requestid	This is the ID of the `<deleteann>` request.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.deleteann.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.deleteann		This indicates that an error occurred while trying to perform the `<deleteann>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying to process this dialog action. • cancelled - This dialog action

Event	Attributes	Description
		has been cancelled.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.remote.done		This event indicates the success of the remote request and that the remote resource has executed the dialog application.
	requestid	This is the ID of the `<remote>` request.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.remote.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.remote		This indicates that an error occurred while trying to perform the `<remote>` request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the

Event	Attributes	Description
		failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying to process this dialog action. • cancelled - This dialog action has been cancelled.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.setdialogdefaultdest.done		This event indicates the success of the setdialogdefaultdest request and that the resource has accepted the default destination.
	requestid	This is the ID of the `<setdialogdefaultdest>` request.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)
dialog.setdialogdefaultdest.requestid		This event provides the application with request ID for the given request that was invoked.
	requestid	This is the ID associated with the request from the orchestration application or the resource.
error.dialog.setdialogdefaultdest		This indicates that an error occurred while trying to perform the `<setdialogdefaultdest>` request.
	requestid	This is the ID associated with the request.

Event	Attributes	Description
	error	This is the type of error that occurred. The following are the possible values: • invalidattribute - The attribute yyy:xxx has an invalid value (zzz) or is not allowed under the conditions of the request. yyy is the name of the element associated with the attribute. xxx is the name of the attribute. zzz is the value of the attribute. • unknown - The cause of the failure is unknown. • invalidstate.state (null, hold, treating, routed) - The interaction is in an invalid state and cannot finish the dialog action. • remote - There was an error in the media server while trying to process this dialog action. • cancelled - This dialog action has been cancelled.
	description	This is a more detailed description of the error.
	extensions	If present may provide addition information sent as extension data from the source performing the dialog operation(Since 8.1.2)

Statistic Interface

This interface provides statistical information to the Orchestration logic.

Functions

_genesys.statistic.sData

This function returns the value of a statistic for a specified object. For example, this function can be used to get the number of interactions waiting in a queue (that is, number of callers ahead of this caller), so that it can be announced to the caller via an IVR application while waiting for a target.

```
value _genesys.statistic.sData(object, statistic)
```

Before you can query a statistic using _genesys.statistic.sData within your SCXML file, you should subscribe to the statistic. Please see subscribe for more details.

However, the following list of predefined statistics can be used without subscribing to it within your SCXML file:

- StatTimeInReadyState
- StatAgentsAvailable
- StatAgentsTotal
- StatAgentsBusy
- StatCallsAnswered
- StatCallsCompleted
- StatExpectedWaitingTime
- StatLoadBalance
- StatAgentsInQueueLogin
- StatAgentsInQueueReady

Parameters:

- **JavaScript object**. Starting with ORS 8.1.400.45, the _genesys.statistic.sData function allows you to directly define statistic parameters in the SCXML routing strategy code via a JavaScript object passed as an input parameter. For information on this enhancement, see Direct Statistic Definition in the *Orchestration 8.1.4 Deployment Guide*.
- **object:** STRING which can be a variable or a constant - This is the name of the object for which the statistic value is requested. The format of this parameter value must use the target formats (for details see the Queue Interface Target Formats section), but there are some exceptions based on the type of statistic (that is, the value of the statistic parameter) being requested. The following describes those exceptions:

- If the statistic parameter value is *CallsDistributed, CallsAnswered, DistributedPercentage, DistributedWaitingTime, NotDistributedPercentage*, or *NotDistributedWaitingTime*, then the object parameter value can only be one of the following:
  - **"R"** - This represents the Queue functional module system as a whole (router).
  - **"RP"** - This is the route point associated with this session (that is for example, `_genesys.ixn.interactions[ixnid].voice.dnis`)
  - In all other cases, it is the name of a virtual queue.
- **statistic:** STRING which can be a variable or a constant - This is the name of the statistic being requested. A value of zero (0) will returned if:
  - The connection to the functional module's statistical system (Stat Server) is not available.
  - If the specified statistic is neither on the list of predefined statistics nor defined in the strategy.
  - If the string defining the object is not of a valid format or the type of object does not support that requested statistic.

Returns: `value`: NUMBER (FLOAT) - This is the current value of the statistic.

The following is an example of using a predefined statistic without subscribing:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:statistic="http://www.genesyslab.com/modules/statistic"
        initial="initial" _persist="false">
  <datamodel>
    <data ID="reqid"/>
  </datamodel>
  <state id="initial" >
    <transition event="interaction.added" target="check" />
  </state>
  <state id="check">
        <transition
cond="_genesys.statistic.sData('SipGr_1@StatServer.GA','StatAgentsAvailable')==1"
target="routing"/>
  </state>

  <state id="routing">
    <onentry>
      <queue:submit requestid="_data.reqid" queue="'802_SipSwitch@.Q'" priority="5"
timeout="20">
        <queue:targets type="dn">
          <queue:target name="'702'" />
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'Queue Submit DONE'"/>
      <log expr="'SDATA for SipGr_1 =
'+_genesys.statistic.sData('SipGr_1@StatServer.GA','StatAgentsAvailable')"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error">
      <log expr="uneval( _event )" />
    </transition>
  </state>
  <final id="exit" />
  <final id="error" />
```

```
</scxml>
```

This is an example of a custom statistic which requires subscription:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:ixn="http://www.genesyslab.com/modules/interaction"
        xmlns:statistic="http://www.genesyslab.com/modules/statistic"
        initial="initial" _persist="false">
  <datamodel>
    <data ID="reqid"/>
    <data ID="ixnid"/>
    <data ID="q1"/>
    <data ID="q2"/>
    <data ID="q3"/>
    <data ID="time_delay" expr="'3s'" />
  </datamodel>
  <state id="initial" >
    <transition event="interaction.added" target="subscribe" />
  </state>

  <state id="subscribe">
    <onentry>
      <log expr="'======== Inside Subscribe ========'"/>
      <statistic:subscribe object="'801_SipSwitch@StatServer.Q'" statistic="'AvgWaitingTime'"
interval="0"/>
      <statistic:subscribe object="'802_SipSwitch@StatServer.Q'" statistic="'AvgWaitingTime'"
interval="0"/>
      <statistic:subscribe object="'803_SipSwitch@StatServer.Q'" statistic="'AvgWaitingTime'"
interval="0"/>
    </onentry>

    <transition event="statistic.subscribe.done" target="delay"/>
    <transition event="error.statistic.subscribe" target="error"/>
  </state>

  <state id="delay">
    <onentry>
      <log expr="'======== Inside Delay ========'"/>
      <send event="'SynchroEvent'" delay="_data.time_delay"/>
    </onentry>
    <transition event="SynchroEvent" target="check" />
  </state>

  <state id="check">
    <onentry>
      <log expr="'Script Start========================================'"/>
      <script>
        _data.q1 = _genesys.statistic.sData('801_SipSwitch@StatServer.Q','AvgWaitingTime');
        _data.q2 = _genesys.statistic.sData('802_SipSwitch@StatServer.Q','AvgWaitingTime');
        _data.q3 = _genesys.statistic.sData('803_SipSwitch@StatServer.Q','AvgWaitingTime');
        __Log('AvgWaitingTime for q1 = '+_data.q1);
        __Log('AvgWaitingTime for q2 = '+_data.q2);
        __Log('AvgWaitingTime for q3 = '+_data.q3);
      </script>
      <log expr="'Script End========================================'"/>
    </onentry>
        <transition
cond="_genesys.statistic.sData('803_SipSwitch@StatServer.Q','AvgWaitingTime') == 0"
target="routing" />
  </state>

  <state id="routing">
```

```
    <onentry>
      <queue:submit requestid="_data.reqid" queue="'802_SipSwitch@.Q'" priority="5"
timeout="20">
        <queue:targets type="dn">
          <queue:target name="'702'" />
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'Queue Submit DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error">
      <log expr="uneval( _event )" />
    </transition>
  </state>
  <final id="exit" />
  <final id="error" />
</scxml>
```

_genesys.statistic.getAvgData

This function calculates the specified statistic for all listed targets and returns the average value of this statistic.

`value _genesys.FMname.getAvgData(objects, statistic)`

Parameters:

- **objects:** STRING which can be a variable or a constant - This parameter is the list of comma-separated objects (targets, in the case of target selection functionality) which this calculation is to be done against.
- **statistic:** STRING which can be a variable or a constant - This parameter defines the statistic that is to be used in this calculation.

Returns: `value`: NUMBER (FLOAT) - The result of the function is the average value for the requested statistic, based on the list of objects and their statistical values.

_genesys.statistic.getMinData

This function calculates the specified statistic for all listed targets and returns the minimum value of this statistic.

`value _genesys.FMname.getMinData(objects, statistic)`

Parameters:

- **objects:** STRING which can be a variable or a constant - This parameter is the list of comma-separated objects (targets, in the case of target selection functionality) which this calculation is to be done against.
- **statistic:** STRING which can be a variable or a constant - This parameter defines the statistic that is to be used in this calculation.

Returns: `value`: NUMBER (FLOAT) - The result of the function is the minimum value for the requested statistic, based on the list of objects and their statistical values.

_genesys.statistic.getMaxData

This function calculates the specified statistic for all listed targets and returns the maximum value of this statistic.

`value _genesys.FMname.getMaxData(objects, statistic)`

Parameters:

- **objects**: STRING which can be a variable or a constant - This parameter is the list of comma-separated objects (targets, in the case of target selection functionality) which this calculation is to be done against.
- **statistic:** STRING which can be a variable or a constant - This parameter defines the statistic that is to be used in this calculation.

Returns: `value`: NUMBER (FLOAT) - The result of the function is the maximum value for the requested statistic, based on the list of objects and their statistical values.

Action Elements

The following are the statistic-specific actions.

<subscribe>

This action allows an application to dynamically subscribe to a particular statistic and object pair for this session.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to

Name	Required	Type	Default Value	Valid Values	Description
					be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
object	true	value expression	none	Any value expression that returns a valid string that follows the target formats (for details see the Queue interface Target Formats section)	A value expression which returns the name of the object name associated with this subscription request. The following is the set of valid object types: • Agent • Agent Group (virtual or real) • Campaign • Campaign Group • Destination Label • Interaction Queue • Place • Place Group • Queue (virtual or real)

Name	Required	Type	Default Value	Valid Values	Description
					• Queue Group • Routing Point (virtual and real) See SCXML Legal Data Values and Value Expressions for details.
statistic	true	value expression	none	Any value expression that returns a valid string that represents a valid statistic	A value expression which returns the name of the statistic associated with this subscription request. Any statistic name can be specified except for the following: • CallsDistributed • CallsAnswered • DistributedPercentage • DistributedWaitingTime • NotDistributedPercentage • NotDistributedWaitingTime See SCXML Legal Data Values and Value Expressions for details.
interval	false	value expression	30	A value expression which returns an integer that is greater than 5	A value expression which returns an integer that represents the number of seconds to wait for the interval. See SCXML Legal Data Values and Value Expressions for details. The integer returned must be interpreted as a time interval when an updated value of the statistic is sent.

Note: There can only be one active subscription for a given object and statistic pair and the session.

If there is an active subscription for an object and statistic pair and another `<subscribe>` action is invoked, the new request will be rejected with an error event. If the developer wants to change the interval of a given subscription, they will have to `<unsubscribe>` and `<subscribe>` again with the new value.

The following is an example:

```
<state id="do_subscribe">
  <datamodel>
        <data id="reqid"/>
  </datamodel>
  <onentry>
        <statistic:subscribe requestid="_data.reqid" object="'1234.Q'"
                statistic="'InVQWaitTime'"/>
  </onentry>
  <transition event="statistic.subscribe.done" target="statex"/>
  <transition event="error.statistic.subscribe" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `statistic.subscribe.done` - This event is sent when the request has been accepted by the system and the statistic subscription has started for this session.
- `error.statistic.subscribe` - This event is sent when the request has failed for some reason.
- `statistic.update` - This event is sent at the end of each time interval while the subscription is active.

<unsubscribe>

This action allows an application to unsubscribe from a particular statistic and object pair for this session.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
requestid	false	location expression	none	Any valid location expression	This is the location for the request ID that is returned as part of this request. Any data model expression evaluating to a data model

Name	Required	Type	Default Value	Valid Values	Description
					location. See SCXML Location Expressions for details. The location's value will be set to an internally generated unique string identifier to be associated with the action being sent. If this attribute is not specified, the event identifier is dropped. This identifier can be tested by the completion event handler to distinguish among several outstanding requests. If this attribute is not specified, the identifier can be acquired from the completion event. Every request must receive a unique identifier.
object	true	value expression	none	Any value expression that returns a valid string that follows the target formats (for details see the [[Queue_Interface#Target_Formats]] section)	A value expression which returns the name of the object name associated with this subscription request. The following is the set of valid object types: • Agent • Agent Group (virtual or real) • Campaign • Campaign Group • Destination Label • Interaction Queue

Name	Required	Type	Default Value	Valid Values	Description
					• Place • Place Group • Queue (virtual or real) • Queue Group • Routing Point (virtual and real) See SCXML Legal Data Values and Value Expressions for details.
statistic	true	value expression	none	Any value expression that returns a valid string that represents a valid statistic	A value expression which returns the name of the statistic associated with this subscription request. Any statistic name can be specified except for the following: • CallsDistributed • CallsAnswered • DistributedPercentage • DistributedWaitingTime • NotDistributedPercentage • NotDistributedWaitingTime See SCXML Legal Data Values and Value Expressions for details.

The following is an example:

```
<state id="do_unsubscribe">
  <datamodel>
        <data id="reqid"/>
  </datamodel>
  <onentry>
        <statistic:unsubscribe requestid="_data.reqid" object="'1234.Q'"
        statistic="'InVQWaitTime'"/>
  </onentry>
  <transition event="statistic.unsubscribe.done" target="statex"/>
  <transition event="error.statistic.unsubscribe" target="statey"/>
</state>
```

Children

None

Events

The following events can be generated as part of this action:

- `statistic.unsubscribe.done` - This event is sent when the request has been accepted by the system and the statistic subscription has been terminated for this session.
- `error.statistic.unsubscribe` - This event is sent when the request has failed for some reason.

Events

The following are the statistic action result events:

Event	Attributes	Description
statistic.subscribe.done		This event indicates the success of the request and that the subscription has been activated for this session.
	requestid	This is the ID associated with the request.
error.statistic.subscribe		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred: • Request Invalid • Subscription Already Active
	description	This is a more detailed description of the error.

Event	Attributes	Description
statistic.unsubscribe.done		This event indicates the success of the request and that the subscription has been terminated for this session.
	requestid	This is the ID associated with the request.
error.statistic.unsubscribe		This indicates that an abnormal condition occurred while trying to perform the request. This event will be sent as a result of a timeout of the request as well as due to problems with the request or interaction itself.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred: • Request Invalid
	description	This is a more detailed description of the error.

The following are the stat asynchronous events:

Event	Attributes	Description
statistic.update		This provides the update value of the statistic.
	object	This is the name of the object that the statistic is associated with. (string)
	statistic	This is the name of statistic. (string)
	value	This is a floating point number which represent the updated value of the statistic that was subscribed to.

Resource Interface

```
This functional module contains enumeration objects that can be used in other functional
modules.  There are currently no events or actions generated by this functional module.
```

Resource Interface

A common entity that used across functional module interfaces is called a resource. A resource is an entity in a business which is involved in helping customers with the services they need. This resource can either be directly involved with the customer through an actual interaction (an IVR, website, knowledge management system, and so on) or be indirectly involved with the customer by doing the processing on behalf of the customer (for example, an agent). A resource can be either a human (for example, an agent) or a device (for example, an IVR). Each resource has a device associated with it to allow it to control the media interactions it is going to help process.

Object Model

_genesys.resource Object

This is the global root object for the Resource functional module interface. This object is maintained by the Resource functional module that implements this interface. The name of the object will be "**_genesys.resource**". There are currently no data properties associated with this object.

_genesys.resource.resourceType ENUM Object

This represents the resource type enumeration. This enumeration is maintained by the orchestration platform. This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
CFGNoDN	read only	integer	none	0	This indicates that no DN type should be used.
CFGExtension	read only	integer	none	1	This indicates that the extension DN type should be used.
CFGACDPosition	read only	integer	none	2	This indicates that the ACD position DN type should be used.
CFGACDQueue	read only	integer	none	3	This indicates that the ACD queue DN type should be used.
CFGRoutingPoint	read only	integer	none	4	This indicates that the

Name	Access	Type	Default Value	Valid Values	Description
					routing point DN type should be used.
CFGVirtACDQueue	read only	integer	none	5	This indicates that the vitural ACD queue DN type should be used.
CFGVirtRoutingPoint	read only	integer	none	6	This indicates that the virtual routing point DN type should be used.
CFGEAPort	read only	integer	none	7	This indicates that the EA port DN type should be used.
CFGVoiceMail	read only	integer	none	8	This indicates that the voice mail DN type should be used
CFGCellular	read only	integer	none	9	This indicates that the cellular DN type should be used.
CFGCP	read only	integer	none	10	This indicates that the CP DN type should be used.
CFGFAX	read only	integer	none	11	This indicates that the FAX DN type should be used.
CFGData	read only	integer	none	12	This indicates that the data DN type should be used.
CFGMusic	read only	integer	none	13	This indicates that the music DN type should be used.
CFGTrunk	read only	integer	none	14	This indicates that the trunk DN type should be used
CFGTrunkGroup	read only	integer	none	15	This indicates that the trunk

Name	Access	Type	Default Value	Valid Values	Description
					group DN type should be used.
CFGTieLine	read only	integer	none	16	This indicates that the tie line DN type should be used.
CFGTieLineGroup	read only	integer	none	17	This indicates that the tie line group DN type should be used.
CFGMixed	read only	integer	none	18	This indicates that the mixed DN type should be used.
CFGExtRoutingPoint	read only	integer	none	19	This indicates that the external routing point DN type should be used.
CFGDestinationLabel	read only	integer	none	20	This indicates that the destination label DN type should be used.
CFGServiceNumber	read only	integer	none	21	This indicates that the service number DN type should be used.
CFGRoutingQueue	read only	integer	none	22	This indicates that the routing queue DN type should be used.
CFGCommunicationDN	read only	integer	none	23	This indicates that the communication DN type should be used.
CFGEmail	read only	integer	none	24	This indicates that the email DN type should be used.
CFGVoIP	read only	integer	none	25	This indicates that the voip

Name	Access	Type	Default Value	Valid Values	Description
					DN type should be used.
CFGVideo	read only	integer	none	26	This indicates that the video DN type should be used.
CFGChat	read only	integer	none	27	This indicates that the chat DN type should be used.
CFGCoBrowse	read only	integer	none	28	This indicates that the cobrowse DN type should be used.
CFGVoIPService	read only	integer	none	29	This indicates that the VoIPService DN type should be used.
CFGWorkflow	read only	integer	none	30	This indicates that the workflow DN type should be used.
any	read only	integer	none	1000	This indicates that any DN type should be used.

_event.data.resource Object

The _event.data.resource object in the queue.submit.done event will contain the resource in the format described here (ready to be used as the value of "to" in action items like redirect, singlesteptransfer, and so on). This is the set of properties for the object:

Name	Access	Type	Default Value	Valid Values	Description
type	r/w	string	none	A, AP, GA, GP, WB, IQ, Q, RP, DN	This is the type of resource being represented. • A - Agent ID • AP - Agent Place ID • GA - Agent Group ID

Name	Access	Type	Default Value	Valid Values	Description
					• GP - Place Group ID • WB - Workbin ID • IQ - Interaction Queue • Q - Queue • RP - Route Point • DN - Directory number
dn	r/w	string	none		This is for voice-related resources and is the DN for the resource. This can be a Queue, Route Point, or Directory Number.
agent	r/w	string	none		This is for voice-related or non-voice-related resources and can be either of the following: • Voice - Agent (A) or Agent Group (GA) • Non-Voice - Agent (A)
place	r/w	string	none		This is for voice-related or non-voice-related resources and can be either of the following: • Voice - Place (AP) or Place Group (GP) • Non-Voice -

Name	Access	Type	Default Value	Valid Values	Description
					Place (AP)
id	r/w	string	none		This is for non-voice-related resources and can be either of the following: • Interaction Queue (IQ) • Workbin (WB)
switch	r/w	string	none		This is for voice-related resources and is associated with the dn property.
vq	r/w	string	none		This is for voice-related or non-voice-related resources and is the virtual queue associated with the resource.
wb_type	r/w	string	none		This is for non-voice-related workbin resources and can be one of the following: • Agent (A) or Agent Group (GA) or Place (AP) or Place Group (GP)
wb_owner	r/w	string	none		This is for non-voice-related workbin resources and is the name of the workbin owner, with the string presenting the name of the configuration layer object that is the owner of the workbin.

General rules for specifying target resources:

- For voice interactions, the **dn** property is mandatory. If the **switch** property is not used, the specified **dn** property will be considered as local (with the same T-Server). All other keys except multimedia-specific ones can provide information about the target the interaction is routed to. This information can be attached to the interaction.
- For multimedia interactions, the **dn** property and the **switch** are ignored. The required information depends on the type of target - if it is an agent, then key **agent** is mandatory. If it is a place, then key **place** is mandatory. If it is an interaction queue or a workbin, then key **id** is mandatory. For workbin, **wb_type and wb_owner** keys are also mandatory.

Samples: To route a voice call to some DN on another switch:

```
<onentry>
  <script>
        var dest = {type:"DN"
                    dn:"702",
                    id:"702_sip",
                    place:"702",
                    'switch':"another_switch"};
  </script>
  <ixn:redirect interactionid="_data.ixnid" from="'RP_sip1'" to="dest"  />
</onentry>
```

To route a multimedia call to an agent :

```
<onentry>
  <script>
        var dest = {type:"A"
                    agent:"702_sip"};
  </script>
  <ixn:redirect interactionid="_data.ixnid" from="'RP_sip1'" to="dest"  />
</onentry>
```

To route a call to a workbin:

```
<onentry>
  <script>
        var dest = {type:"WB",
                    id:"WorkbinTypeName",
                    wb_type:"GA",
                    wb_owner:"SomeAgentGroup"};
  </script>
  <ixn:redirect interactionid="_data.ixnid" from="'RP_sip1'" to="dest"  />
</onentry>
```

To route a call to a persistent queue:

```
<onentry>
  <script>
        var dest = {type:"IQ",
                    id:"InteractionQueueName"};
  </script>
  <ixn:redirect interactionid="_data.ixnid" from="'RP_sip1'" to="dest"  />
</onentry>
<script> dest= {"type"="IQ", "id"="InteractionQueueName"}; </script>
<redirect ...from="12345" to="dest"/>
```

When making a redirect, transfer, and so on, if the target contains additional information (vq, agent, place, sometimes id) before routing, the corresponding data should be attached to the interaction (in the same way URS currently does).

Functions

There are none at this time.

Action Elements

There are none at this time.

Events

There are none at this time.

Elasticsearch Connector

Starting with 8.1.400.58, ORS enhances its Elasticsearch real-time reporting capabilities. Enhancments include a new Orchestration SCXML extension, elasticconnector. The purpose of `elasticconnector` SCXML extension is to:

- Remove management of connectivity to Elasticsearch cluster from the SCXML strategy.
- Provide simplified access to Elasticsearch APIs.

elasticconnector Action Elements

ORS supports the following new Action elements that can be used with Composer's SCXML State block:

<createindextemplate>

- Execution of this Action results in a request to Elasticsearch only if the current ORS did not execute such action with the same template name and the same or higher order.
- If the pattern in the “index” attribute matches the “performance” daily index (like “perf*”, “performance-*”, and so on), the action will fail with the appropriate error and error description.
- If the pattern in the “index” attribute matches the “session” daily index and “order” is defined as 0, action will fail with the appropriate error and error description. As a general rule – the template always has order=0 and only one template is created per “session” and “performance” indexes; custom templates always have to have order>0.

[+] createindextemplate

Name	Required	Type	Def. value	Valid values	Description
requestid	False	Location expression	none	Any valid location expression, which represents a string.	This is the location for the request ID that is returned as part of this request. Standard attribute of all Orchestration actions.
name	True	Value expression	none	Any value expression that returns a valid string.	template name
index	True	Value expression	none	Any value expression that returns a valid string.	Elasticsearch index name (pattern).

order	False	Value expression	0	Any value expression that returns a valid integer.	Elasticsearch template order.
type	True	Value expression	none	Any value expression that returns a valid string.	Elasticsearch type name.
mapping	True	Value expression	none	Any valid ECMAScript object.	Object that represents the whole body of Elasticsearch property of `mappings.type` in the mapping request.
timeout	False	Value expression	0	Any value expression that returns a valid integer.	The integer returned must be interpreted as a time interval in milliseconds. This interval begins when action is executed. A failed and timed out fetch returns the `error.elasticconnector.createindex event`.

<createdoc>

Execution of this Action results in an add document request from ORS to Elasticsearch (via the index API).

[+] createdoc

Name	Required	Type	Def. value	Valid values	Description
requestid	false	Location expression	none	Any valid location expression which represents a string.	This is the location for the request ID that is returned as part of this request. Standard attribute of all Orchestration actions.
id	false	Value expression	none	Any value expression that returns a valid string	Elasticsearch document within an index. If not specified, an ID will be automatically generated. Explicit ID assignment and automatic ID generation cannot be mixed within same index.
index	true	Value expression	none	Any value expression that returns a valid string.	Elasticsearch index name

type	true	Value expression	none	Any value expression that returns a valid string.	Elasticsearch type name
request	true	Value expression	none	Any valid ECMAScript object	Object that represents the whole body of an Elasticsearch create document API request.
bulk	false	Boolean expression	false	true,false	If true, Elasticsearch request will be just added to the ORS bulk request buffer and the `elasticconnector.createdoc.done` event will be raised immediately. If false, a request to create document will be sent to Elasticsearch and the corresponding event and `elasticconnector.createdoc.done` or `error.elasticconnector.createdoc` will be raised after a response from Elasticsearch.
timeout	false	Value expression	0	Any value expression that returns a valid integer.	The integer returned must be interpreted as a time interval in milliseconds. This interval begins when action is executed. A failed and timed out fetch returns the `error.elasticconnector.createdoc` event.
settings	false	Value expression	none	Any valid ECMA script object.	Object that represents the body of the Elasticsearch /_settings index API request.

The `settings` attribute has been introduced in version 8.1.400.64. For more information, see ORS Options for Elasticsearch 5.3, Index Settings for Custom Indexes section.

<updatedoc>

Execution of this Action results in an update document request from ORS to Elasticsearch (via update API).

[+] updatedoc

Name	Required	Type	Def. value	Valid values	Description
requestid	false	Location expression	none	Any valid location expression which represents a string.	This is the location for the request ID that is returned as part of this request. Standard attribute of all Orchestration actions.

id	true	Value expression	none	Any value expression that returns a valid string.	Elasticsearch document ID within an index.
index	false	Value expression	Current session daily index.	Any value expression that returns a valid string.	Elasticsearch index name. If not specified, current session daily index will be used. Also, in this case, the type attribute must be not specified, or the action will fail.
type	false	Value expression	“session”	Any value expression that returns a valid string.	Elasticsearch type name. If not specified, the “index” attribute should be not specified as well and “session” type will be used.
request	true	Value expression	none	Any valid ECMAScript object	Object that represents the whole body of an Elasticsearch update document API request.
bulk	false	Boolean expression	true	true,false	If true, Elasticsearch request will be just added to the ORS bulk request buffer and elasticconnector.updatedoc.done event will be raised immediately. If false, a request to update document will be sent to Elasticsearch and corresponding event `elasticconnector.updatedoc.done` or `error.elasticconnector.updatedoc` will be raised after response from Elasticsearch.
timeout	false	Value expression	0	Any value expression that returns a valid integer.	The integer returned is interpreted as a time interval in milliseconds. This interval begins when the action is executed. A failed and timed out fetch returns the `error.elasticconnector.updatedoc` event.
settings	false	Value expression	none	Any valid ECMA script object.	Object that represents the body of the Elasticsearch /_settings index API request.

The `settings` attribute has been introduced in version 8.1.400.64. For more information, see ORS Options for Elasticsearch 5.3, Index Settings for Custom Indexes section.

<deletedoc>

Execution of this Action results in a delete document request from ORS to Elasticsearch (via delete API).

[+] deletedoc

Name	Required	Type	Def. value	Valid values	Description
requestid	false	Location expression	none	Any valid location expression which represents a string.	This is the location for the request ID that is returned as part of this request. Standard attribute of all Orchestration actions.
id	true	Value expression	none	Any value expression that returns a valid string.	Elasticsearch document ID within an index.
index	true	Value expression	none	Any value expression that returns a valid string.	Elasticsearch index name
type	true	Value expression	none	Any value expression that returns a valid string.	Elasticsearch type name
bulk	true	Boolean expression	true	true,false	If true, Elasticsearch request will be added to ORS bulk request buffer and `elasticconnector.deletedoc.done` will be raised immediately. If false, request to delete document will be sent to Elasticsearch and corresponding `elasticconnector.deletedoc.done` or `error.elasticconnector.deletedoc` event will be raised after a response from Elasticsearch.
timeout	false	Value expression	0	Any value expression that returns a valid integer.	The integer returned is interpreted as a time interval in milliseconds. This interval begins when the action is executed. A failed and timed out fetch returns the error.elasticconnector.deletedoc event.

Events

The following Events are supported:

[+] Events

}

Name	Attributes	Description
elasticconnector.createindextemplate.done		This event indicates the success of the request.
	requestid	This is the ID of the request.
error.elasticconnector. createindextemplate		This indicates that an error occurred while trying to perform the createindextemplate request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following is a specific error code: protocol.errorcode - This represents the protocol-specific errors that occur when the attempting the request.
	description	This is a more detailed description of the error
elasticconnector.createdoc.done		This event indicates the success of the request.
	id	This is ID of created document.
	requestid	This is the ID of the request.
error.elasticconnector.createdoc		This indicates that an error occurred while trying to perform the createdoc request.
	requestid	This is the ID associated with the request.
	error	This is the type of error that occurred. The following is a specific error code: protocol.errorcode - This represents the protocol-specific errors that occur when the attempting the request.
	description	This is a more detailed description of the error

elasticconnector.updatedoc.done		This event indicates the success of the request.
	Requestid	This is the ID of the request.
error.elasticconnector.updatedoc		This indicates that an error occurred while trying to perform the updatedoc request.
	Requestid	This is the ID associated with the request.
	Error	This is the type of error that occurred. The following is a specific error code: protocol.errorcode - This represents the protocol-specific errors that occur when the attempting the request.
	Description	This is a more detailed description of the error
elasticconnector.deletedoc.done		This event indicates the success of the request.
	Requestid	This is the ID of the request.
error.elasticconnector.deletedoc		This indicates that an error occurred while trying to perform the deletedoc request.
	Requestid	This is the ID associated with the request.
	Error	This is the type of error that occurred. The following is a specific error code: protocol.errorcode - This represents the protocol-specific errors that occur when the attempting the request.
	Description	This is a more detailed description of the error

Agent Extension

Beginning with release 8.1.400.81, ORS introduces a new SCXML extension, **Agent**, which can be used for implementing agent management features like logout, DND, and ability to change agent state for voice media.

> **Important**
> Currently, only voice interactions are supported. The only allowed value for the `media` property is `_genesys.ixn.mediaType.TMediaVoice`.

All actions of the Agent functional module are finalized as soon as T-Library accepts a request, that is, the action done-event is sent to the SCXML session as soon as the request is successfully sent to T-Server.

agent Action Elements

ORS supports the following new Action elements that can be used with Composer's SCXML State block:

> **Important**
> To use these agent actions, you must manually configure the corresponding namespace in Composer.

<logout>

This action logs out the agent from the ACD group. This is equivalent to the `TAgentLogout` request in T-Server.

Name	Required	Type	Def. value	Valid values	Description
requestid	False	Location expression	None	Any valid location expression, which represents a string.	This is the location for the request ID that is returned as part of this request. Standard attribute of all Orchestration actions.

Name	Required	Type	Def. value	Valid values	Description
agent	False	Value expression	None	Any valid expression, which represents a string.	Agent ID
for	True	Value expression	None	Any valid expression, which represents an object.	The object structure must be as follows: `{ “dn”:<DN number>,“switch”:<Switch name><br> }`<br>**Note**: The dn property is mandatory.
hints	False	Value expression	None	Any valid ECMA script object.	The object structure must be as follows: `{ “extensions”:<object>,“reaso }`

<setmediastate>

This action sets the state of the agent as ready or not ready to receive the call. This is equivalent to the `TAgentSetReady` or `TAgentSetNotReady` requests in T-Server.

Name	Required	Type	Def. value	Valid values	Description
requestid	False	Location expression	None	Any valid location expression, which represents a string.	This is the location for the request ID that is returned as part of this request. Standard attribute of all Orchestration actions.
agent	False	Value expression	None	Any valid expression, which represents a string.	Agent ID
media	True	Value	None	`_genesys.ixn.medi[illegible]aVoice`	The type of Medi[illegible]

Name	Required	Type	Def. value	Valid values	Description

		expression			contain objects of the following structure:
					`{`
					`“media”:_genesys.ixn.mediaTy`
					`“state”:<”ready” or “notready”>`
					`}`

for	True	Value expression	None	Any valid expression, which represents an object.	The object structure must be as follows: `{ “dn”:<DN number>,“switch”:<Switch name><br> }`<br>**Note**: The dn property is mandatory.

hints	False	Value expression	None	Any valid ECMA script object.	The object structure must be as follows:
					`{`
					`“extensions”:<object>,“reaso`
					`}`

<setdnd>

This action sets the **Do-Not-Disturb** (DND) feature to *On* or *Off* for the telephony object. This is equivalent to the `TSetDNDOn` or `TSetDNDOff` requests in T-Server.

Name	Required	Type	Def. value	Valid values	Description
requestid	False	Location expression	None	Any valid location expression, which represents a string.	This is the location for the request ID that is returned as part of this request. Standard attribute of all Orchestration actions.

Name	Required	Type	Def. value	Valid values	Description
agent	False	Value expression	None	Any valid expression, which represents a string.	Agent ID
set	True	Boolean value expression	None	Any boolean expression that returns a true or false.	Sets the desired DND state. If true, DNDOn, else, DNDOff.
for	True	Value expression	None	Any valid expression, which represents an object.	The object structure must be as follows: `{ “dn”:<DN number>,“switch”:<Switch name> }` **Note**: The dn property is mandatory.
hints	False	Value expression	None	Any valid ECMA script object.	The object structure must be as follows: `{ “extensions”:<object>,“reas }`

Events

The following events are supported by the Agent extension:

Event Name	Attributes	Description
agent.logout.done		This event indicates the success of the request.
	requestid	This is the ID of the request.
error.agent.logout		This event indicates that an abnormal condition occurred while trying to perform the request.
	requestid	This is the ID of the request.

Event Name	Attributes	Description
	error	This is the type of error that occurred. The following are the possible values: • unknown • invalidresource • servererror
	description	This is a more detailed description of the error.
agent.setmediastate.done		This event indicates the success of the request.
	requestid	This is the ID of the request.
error.agent.setmediastate		This error indicates that an abnormal condition occurred while trying to perform this request.
	requestid	This is the ID of the request.
	error	This is the type of error that occurred. The following are the possible values: • unknown • invalidresource • servererror
	description	This is a more detailed description of the error.
agent.setdnd.done		This event indicates the success of the request.
	requestid	This is the ID of the request.
error.agent.setdnd		This error indicates that an abnormal condition occurred while trying to perform this request.
	requestid	This is the ID of the request.
	error	This is the type of error that occurred. The following are the possible values: • unknown • invalidresource • servererror
	description	This is a more detailed description of the error.

Migration from IRD

The following table documents the mapping from URS/IRD function block functionality to SCXML and Functional Modules functionality for use in Orchestration. This mapping will result in the creation of an SCXML snippet or a specific `<state>` element definition which represents the functionality of the function block. The following is a `<state>` element template that could be used:

```
<state id="function_block_name">
        <datamodel>
                <! - This defines the data needed for processing this function block -->
        </datamodel>
        <onentry>
                <! - Do the necessary function block action set up and action execution -->
                <xxxx:yyyy requestid="_data.reqid" .../>
        </onentry>
        <transition event="xxxx.yyyy.done" cond="..." target="...">
                <! - transition based on results (equal to function block output port) -->
        </transition>
        <transition event="xxxx.zzzz" cond="..." target="...">
                <! - additional actions if necessary and transitions based on it. (equal to
function block output port ) -->
        </transition>
        <transition event="error.xxxx.yyyy">
                <! - transition based on results - (equal to function block error port) -->
        </transition>
</state>
```

Function Block	SCXML and Functional Module equivalent
Acknowledgement	`<msgbased:createmessage>` for appropriate media
Add Record	`<session:fetch..." method = "'post'" srcexpr="'http://server.com...req=AddRecord'">`
ANI	`<script>` if done in orchestration, `<session:fetch>` with application server code, if done outside.
Assign	`<assign>` using ECMAScript object properties and functions
Attach Categories	Use _genesys.ixn.interactions[].udata with category information (Business Attribute CME objects) from configuration server
Autoresponse	`<msgbased:createmessage>` for appropriate media
Business	`<script>` if done in orchestration, `<session:fetch>` with application server code, if done outside.
Busy	`<dialog:playsound>`

Function Block	SCXML and Functional Module equivalent
Call Subroutine	`<include>` or `<invoke>`
Cancel Call	`<dialog:stop>`
Chat Transcript	`<msgbased:createmessage ... chattranscript="'xxx'">` for appropriate media
Classify	`<classification:classify>`
Classify (segmentation)	`<classification:classify>`, `<script>` if done in orchestration, `<session:fetch>` with application server code, if done outside.
Collect Digits	`<dialog:collect>`
Comment	`<log> NOTE: As of Orchestration verion 8.1.200.46 - the URS messages 22001 to 22020 are supported.`
Create Interaction	`<session:fetch srcexpr="'http://server1.com/.../customers/${customer_id}/interaction` `method="'post'" ...>`
Create Email Out	`<msgbased:createmessage>` for appropriate media
Create Notification	`<msgbased:createmessage>` for appropriate media
Create SMS	`<msgbased:createmessage>` for appropriate media
Database Wizard	`<session:fetch>`, specialized application server code
Date	`<script>` if done in orchestration (use ECMAScript standard date functions/objects or Session functional module ECMAScript date and time functions), `<session:fetch>` with application server code, if done outside.
Day of Week	`<script>` if done in orchestration (use ECMAScript standard date functions/objects or Session functional module ECMAScript date and time functions), `<session:fetch>` with application server code, if done outside.
Default	`<queue:default>`
Delete User Announcement	`<dialog:deleteann>`

Function Block	SCXML and Functional Module equivalent
DNIS	`<script>` if done in orchestration, `<session:fetch>` with application server code,
Do Not Call	`<session:fetch..." method = "'post'" srcexpr="'http://server.com...req=DoNotCall'" />`
Entry	`<initial>` or `<scxml initial attribute>`
Error Segmentation	`<script>` if done in orchestration, `<session:fetch>` with application server code, if done outside.
Exit	`<final>`
External Service	`<session:fetch>`, specialized application server code
Fast Busy	`<dialog:playsound>`
Force Routing	`<ixn:redirect>`
Forward E-Mail	`<msgbased:createmessage>` for appropriate media
Function	`<script>` any functional module interface action or ECMAScript function.
Generic	`<script>` if done in orchestration, `<session:fetch>` with application server code, if done outside.
Identify Contact	`<session:fetch srcexpr="'http://server1.com/cv/profiles'" method="get" ... >`
If	`<if>`
IVR	`<dialog:remote>`
Load Balancing	`<queue:submit>`
Macro	`<include>`
MultiAssign	`<assign using ECMAScript object properties and functions>`

Function Block	SCXML and Functional Module equivalent
MultiAttach	`<script>` or `<assign>` using _genesys.FMname.interactions[x].udata object properties
MultiScreen	`<classification:screen>`
Music	`<dialog:playsound>`
Pause	`<runttreatments>` and `<pause>`
Percentage	`<queue:submit>`
Play Announcement	`<dialog:play>` and `<prompts type="tts">`
Play Announcement and collect digits	`<dialog:playandcollect>`
Play Application	`<dialog:start>`
Processed	`<session:fetch..." method = "'post'" srcexpr="'http://server.com...req=RecordProcessed'" />`
Queue Interaction	`<ixn:redirect>`
RAN	`<dialog:playsound>`
Record User Announcement	`<dialog:recordann>`
Redirect E-Mail	`<msgbased:createmessage>` for appropriate media
Reply E-Mail From External Resource	`<msgbased:createmessage>` for appropriate media
Reschedule	`<session:fetch..." method = "'post'" srcexpr="'http://server.com...req=RecordReschedule'" />`
Ringback	`<dialog:playsound>`
Route Interaction	`<queue:submit>`
Screen	`<classification:screen>`

Function Block	SCXML and Functional Module equivalent
Screen (segmentation)	`<classification:screen>`, `<script>` if done in orchestration, `<session:fetch>` with application server code, if done outside.
Selection (target selection)	`<queue:submit>`, `<dialog:runtreatments>`
Send E-Mail	`<msgbased:sendmessage>`
Service level	`<queue:submit>`, `<dialog:runtreatments>`
Set Default Destination	`<dialog:setdialogdefaultdest>`
Silence	`<dialog:playsound>`
Statistics	`<queue:submit>`, `<dialog:runtreatments>`
Stop Interaction	`<ixn:terminate>`
Switch to Strategy	`<include>` or `<invoke>`
Text to Speech	`<dialog:play><prompts type="tts">`
Text to Speech and Collect Digits	`<dialog:playandcollect><prompts type="tts">`
Time	`<script>` if done in orchestration (use ECMAScript standard date functions/objects or Session functional module ECMAScript date and time functions), `<session:fetch>` with application server code, if done outside.
Update Contact	`<session:fetch srcexpr="'http://server1.com/cv/ profiles'" method="put"... >`
Update Record	`<session:fetch..." method = "'post'" srcexpr="'http://server.com...req=UpdateCallCompletionStats'" />`
Verify Digits	`<dialog:playandverify>`
Web Service	`<session:fetch>` with application server code
Workbin	`<ixn:redirect>`

Function Block	SCXML and Functional Module equivalent
Workforce	`<queue:submit>`, `<dialog:runtreatments>`

Orchestration Server Integration

Introduction

Orchestrations server integration allows SCXML applications to reach out and interact with other systems within your enterprise and may not only be used for customer specific related integrations but is also leveraged to support integrations to other Genesys products and components that may not as yet have native actions support, such as Context Services and Outbound Contact Server. Such integrations with Genesys components are briefly described in the Genesys Servers section below. Outward integrations from Orchestration are facilitated by the Orchestration Core Extension <fetch> which is defined in more detail in Core Extensions <fetch> which can be consulted to aid in your custom integration.

Genesys Servers

To facilitate easier use of these Genesys-related Servers, Composer supports specific blocks that may remove the need for custom integration code. Because of this, we recommend that you refer to the Composer *Routing Applications User's Guide* for the latest available blocks. This document can be obtained from the Composer page on this wiki.

Context Services

Context Services is a data repository that provides real-time and historic customer-centric data to SCXML applications through an interface that allows the application users to make important business decisions. For example, it allows you to determine whether a customer should be offered a given service or that the customer is a high value client, or if there are existing SCXML application sessions are already running for this customer. Context Services exposes a set of RESTful APIs to access the customer context data. Refer to Context Services for more information about the RESTFul interface and Context Services. The following is an example of how you would use this REST APIs from and SCXML application - details on CS REST API . `<!-- This is the IdentifyByPhoneNumber CS API request. -->`

```
        <state id="IdentifyByPhoneNumber">
            <onentry>
                <if cond="_data.context_management_services_url == undefined ||
_data.context_management_services_url == ">
                    <raise event="servererror">
                       <param name="description"
expr="'context_management_services_url property not configured'" />
                    </raise>
                <else/>
                    <script>
                        _data.CustomerCount = 0;
                        var includeProfile = "no";
                        var includeExtension = "unique";
                        includeProfile="no";
                        includeExtension="unique";
                    </script>
                    <session:fetch requestid="_data.requestid"
                               srcexpr="_data.context_management_services_url +
'/profiles'"
                               method="'get'" type="'application/json'">
                            <param name="include_profile" expr="includeProfile"/>
                            <param name="include_extensions"
expr="includeExtension"/>
                            <param name="PhoneNumber" expr="ANI"/>
                    </session:fetch>
                </if>
              </onentry>

            <transition event="error.session.fetch" target="Exit_Error">
                <log expr="'Error ' + _event.data.error + ':' +
_event.data.description" />
            </transition>

            <transition event="session.fetch.done" cond="_event.data.content =="
target="Exit_Error">
                <assign location="CustomerCount" expr="0" />
                <log expr="'Error No customer found'" />
            </transition>

            <transition event="session.fetch.done" target="Exit_final">
                <script>
                    var _data.CustomerData = _event.data.content !=  ? eval('(' +
_event.data.content + ')') : new Array();
                    if (_data.CustomerData.length == 1) {
                        if (App_IdentifyByPhoneNumber_IncludeExtension=="unique") {
                            _data.CustomerData = [{'customer_id' :
_data.CustomerData[0].customer_id}];
                        }
                    }
                </script>
                <log expr="'IdentifyByPhoneNumber: ' + CustomerData.length + '
Customer record(s) found'"/>
            </transition>

        </state>
```

Outbound Contact Campaigns

The campaign calling list record control actions will be handled through the `<fetch>` action. The Outbound Contact Server campaign-related HTTP APIs will be mapped to the `<fetch>` attributes and child elements. See OCS Support for HTTP Protocol in the *Outbound Contact Reference Manual*.

Important `<fetch>`-related usage notes with the Outbound Web 2.0 APIs:

- The method attribute value is always "post".
- The enctype attribute value is always "application/json".
- The type attribute value is always "application/json".
- The `<param>` name attribute is always "record".

The following is the general mapping to `<fetch>`, while the sub-sections are detailed mappings and examples for the functions that will be supported.

```
<session:fetch requestid="_data.reqid"
                srcexpr="'http://cvserver1.com/<resource>/<id>?req=actionx'"
                method="post" type="application/json" enctype="application/json">
        <param name="record" expr="_data.rmyrecord"/>
</session:fetch>
```

`<resource>` can be:

- records - This contains the campaign records:
  - `<id>` - This will be the ID of the record.
- phones - This is the phone number associated with a record or set of records:
  - `<id>` - This will be the phone number.
- customer_ids - This is the customer ID associated with a record or set of records:
  - `<id>` - This will be the customer ID.

Adding a New Record

This action adds a new record to an existing campaign's call list. This action covers the "Add_Record" IRD function block. Elements:

- `Req = AddRecord`
- `<resource> = records`
- `<id> = recordid`

The following is an example of how to use the `<fetch>` element to add a new outbound record.

```
<onentry>
        <script>
                Record = new Object();
                Record.GSW_PHONE = "567567567545656";
                Record.GSW_TZ_NAME = "'PST";
                Record.GSW_CALL_RESULT = 28;
                Record.STATUS_CODE = "New";
                Record.CUSTOMER_STATUS = 5;
                Record.GSW_CAMPAIGN = "New Productx";
        </script>

        <session:fetch
                        srcexpr="http://server1.genesyslab.com:8080/records/?req=AddRecord"
                        method="post" enctype=" application/json" type=" application/json">
                <param name="record" expr="local.Record"/>
        </session:fetch>
</onentry>
```

Updating an Existing Record

This action updates an existing record in an existing campaign's call list. This action covers the "Update_Record" and "Reschedule" IRD function blocks. Elements:

- `req = RecordReschedule` or `UpdateCallCompletionStats` or `RecordReject` or `RequestRecordCancel`
- `<resource> = records`
- `<id> = recordid`

The following is an example of how to use the `<fetch>` element to reschedule a record.

```
<onentry>
        <script>
                Record = new Object();
                Record.GSW_DATE_TIME = "10/12/2009";
                Record.GSW_CAMPAIGN = "New Productx";
        </script>

        <session:fetch
                        srcexpr="http://server1.genesyslab.com:8080/records/
123456?req=RecordReschedule"
                        method="post" enctype=" application/json" type=" application/json">
                <param name="record" expr="local.Record"/>
        </session:fetch>
</onentry>
```

The following is an example of how to use the `<fetch>` element to UpdateCallCompletionStats for a record.

```
<onentry>
        <script>
                Record = new Object();
                Record.GSW_CAMPAIGN = "New Productx";
        </script>

        <session:fetch
                        srcexpr="http://server1.genesyslab.com:8080/records/
123456?req=UpdateCallCompletionStats"
                        method="post" enctype=" application/json" type=" application/json">
                <param name="record" expr="local.Record"/>
        </session:fetch>
</onentry>
```

The following is an example of how to use the `<fetch>` element to RecordReject a record.

```
<onentry>
        <script>
                Record = new Object();
                Record.GSW_CALLING_LIST = "First List";
                Record.GSW_CAMPAIGN = "New Productx";
        </script>

        <session:fetch
                        srcexpr="http://server1.genesyslab.com:8080/records/
123456?req=RecordReject"
                        method="post" enctype=" application/json" type=" application/json">
                <param name="record" expr="local.Record"/>
        </session:fetch>
</onentry>
```

Reschedule an Existing Record

This action reschedules an existing record in an existing campaign's call list. This action covers the "Reschedule" IRD function blocks. Elements:

- `req = RecordReschedule`
- `<resource> = records`
- `<id> = recordid`

The following is an example of how to use the `<fetch>` element to reschedule a record.

```
        <onentry>
        <script>
```

```
                Record = new Object();
                Record.GSW_DATE_TIME = "10/12/2009";
                Record.GSW_CAMPAIGN = "New Productx";
        </script>
        <session:fetch
                        srcexpr="http://server1.genesyslab.com:8080/records/
123456?req=RecordReschedule"
                        method="post" enctype=" application/json" type=" application/json">
                <param name="record" expr="local.Record"/>
        </session:fetch>
</onentry>
```

Reject an Existing Record

This action rejects an existing record in an existing campaign's call list. Elements:

- `req = RecordReject`
- `<resource> = records`
- `<id> = recordid`

The following is an example of how to use the `<fetch>` element to RecordReject a record.

```
<onentry>
        <script>
                Record = new Object();
                Record.GSW_CALLING_LIST = "First List";
                Record.GSW_CAMPAIGN = "New Productx";
        </script>
        <session:fetch srcexpr="http://server1.genesyslab.com:8080/records/
123456?req=RecordReject"
                        method="post" enctype=" application/json" type=" application/json">
                <param name="record" expr="local.Record"/v
        </session:fetch>
</onentry>
```

Complete an Existing Record

This action completes an existing record in an existing campaign's call list. It covers the "Processed" IRD function block. Elements:

- `req = RecordProcessed`
- `<resource> = records`
- `<id> = recorded`

The following is an example of how to use the `<fetch>` element to RequestRecordCancel a record.

```
<onentry>
        <session:fetch
        srcexpr="http://server1.genesyslab.com:8080/records/123456?req=RequestRecordCancel"
                method="post" enctype=" application/json" type=" application/json"/>
</onentry>
```

Add the Customer to Do Not Contact List

This action completes an existing record in an existing campaign's call list and adds it to the do not contact list. This action covers the "Do Not Call" IRD function block. Elements:

- `<code>req = DoNotCall</code>`
- `<resource> = records`
- `<id> = recordid`

The following is an example of how to use the `<fetch>` element to RecordProcessed a record.

```
<onentry>
        <session:fetch
                srcexpr="http://server1.genesyslab.com:8080/records/123456?req=DoNotcall"
                method="post" enctype=" application/json" type=" application/json"/>
</onentry>
```

Orchestration Getting Started Guide

Introduction

The aim of the guide is to help you build your SCXML applications. It is assumed at this point that you have installed Orchestration and have it working with other Genesys products. You may also want to install RestClient to test your applications. It is also assumed that you have a general understanding of the Genesys Product Suite, SCXML, as well as Internet technologies such as HTTP, XML, and JSON. If you want to review basic SCXML concepts before continuing, you may find them here.

Writing your first application

Now that you have familiarized yourself with states and transitions, you are ready to write your first application. Let's begin with a simple application that plays music when we receive a voice call:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="begin">
  <datamodel>
        <data id="ixnid" expr="''" />
        <data id="reqid" expr="''" />
  </datamodel>
  <state id="begin">
        <transition event="interaction.added" target="play_music">
                <script>
                        _data.ixnid = _event.data.interactionid;
                </script>
        </transition>
  </state>
  <state id="play_music">
        <onentry>
                <dialog:playsound interactionid="_data.ixnid"
                                  requestid="_data.reqid"
                                  type="'music'"
                                  resource="'music/on_hold'"
                                  duration="'10 '"/>
        </onentry>
        <transition event="dialog.playsound.done" target="exit"/>
        <transition event="error.dialog.playsound" target="error"/>
  </state>

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

Let's look at how this SCXML file works:

- At the top of the file you have included one of the custom ORS extensions, the dialog FM with `xmlns:dialog="www.genesyslab.com/modules/dialog"`.
- The document declares an initial state of `begin`, which is the entry point into the state machine.
- Before we enter the state machine, there is a <datamodel> element which encapsulates any number of <data> elements. This is the single globally visible data model for the entire state machine.
- Each <data> element defines a named data element and is created when the document is loaded.
- While inside the `begin` state, it waits for the `interaction.added` event to trigger a transition.
- The `interaction.added` event is generated when a new interaction is associated with the session. In this case, a voice call will trigger the `interation.added` event which will cause the state machine to transition to the `play_music` state.
- When the transition is triggered, the `executable content` contained in the <script> is executed and the variable `_data.ixnid` within the data model is updated with the interaction id that was returned as part of the _event.data object.
- Once the state `play_music` is entered, the `executable content` contained in the <onentry> is

immediately executed which plays the music file found at `music/on_hold` for a during of 10 seconds.

- <dialog:playsound> is a custom action whose local name is `playsound` and is bound to the namespace `www.genesyslab.com/modules/dialog`.
- The custom action `playsound` has been defined within ORS as an extension. For details, see the section on the dialog interface.
- If 10 seconds of music was played successfully, the `dialog.playsound.done` event is received. Otherwise, we get the `error.dialog.playsound` event. One of these two events will trigger a transition to a final state.
- The final state indicates that the state machine has run to completion.

Now we will add to the scenario by routing the call to an agent after playing music for 10 seconds:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="begin">
  <datamodel>
        <data id="ixnid" expr="''" />
        <data id="reqid" expr="''" />
  </datamodel>
  <state id="begin">
        <transition event="interaction.added" target="play_music">
                <script>
                        _data.ixnid = _event.data.interactionid;
                </script>
        </transition>
  </state>
  <state id="play_music">
        <onentry>
                <dialog:playsound interactionid="_data.ixnid"
                                  requestid="_data.reqid"
                                  type="'music'"
                                  resource="'music/on_hold'"
                                  duration="'10 '"/>
        </onentry>
        <transition event="dialog.playsound.done" target="route_to_agent"/>
        <transition event="error.dialog.playsound" target="error"/>
  </state>
  <state id="route_to_agent">
        <onentry>
                <queue:submit requestid="_data.reqid" interactionid="_data.ixnid"
priority="5" timeout="20">
                        <queue:targets type="agent">
                                <queue:target name="'702_sip'"/>
                        </queue:targets>
                </queue:submit>
        </onentry>

        <transition event="queue.submit.done" target="exit">
                <log expr="'DONE'"/>
                <log expr="_event.data.targetselected"/>
        </transition>
        <transition event="error.queue.submit" target="error">
                <log expr="'ERROR'"/>
        </transition>
  </state>

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

```
</scxml>
```

- First, we added the queue FM at the beginning of the file with `xmlns:queue="www.genesyslab.com/modules/queue"`.
- After playing music for 10 seconds, the `dialog.playsound.done` is received and will trigger a transition to the state `route_to_agent`.
- Once the state `route_to_agent` is entered, the `executable content` contained in the <onentry> is immediately executed which tries to route the call to agent `702_sip`.
- <queue:submit> is a custom action whose local name is `submit` and is bound to the namespace `www.genesyslab.com/modules/queue`.
- The custom action `submit` has been defined within ORS as an extension. For details, see the section on the queue submit.
- If the interaction has been routed successfully to agent `702_sip`, the `queue.submit.done` event is received. Otherwise, we get the `error.queue.submit` event if the interaction was not routed within the 20 seconds timeout period. One of these two events will trigger a transition to a final state.
- Before transitioning to the final `exit` state, the standard action of <log> is called which outputs a string containing information about the <queue:submit> request.

So far, our example has been fairly simple, where a voice call comes in, we play music to it for 10 seconds, then try for 20 seconds to route the call to an agent. But what if the agent is on a call and is unavailable? A more realistic scenario is to wait for the agent to become available and play music to the caller while they are waiting. Of course we don't want to wait indefinitely so let's try for 5 minutes and if the agent doesn't become available, we exit the state machine, as follows:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="begin">
  <datamodel>
        <data id="reqid" expr="''" />
        <data id="ixnid" expr="''" />
  </datamodel>

  <state id="begin">
        <transition event="interaction.added" target="routingwithdialog">
                <script>
                        _data.ixnid = _event.data.interactionid;
                </script>
        </transition>
  </state>

  <parallel id="routingwithdialog">

        <state id="play_music">
                <onentry>
                        <dialog:playsound type="'music'" resource="'music/on_hold'"
duration="'300'"/>
                </onentry>
                <transition event="dialog.playsound.done" target="exit"/>
                <transition event="error.dialog.playsound" target="error"/>
        </state>
        <state id="route_to_agent">
                <onentry>
                        <queue:submit requestid="_data.reqid" interactionid="_data.ixnid"
priority="5" timeout="300">
                                <queue:targets>
```

```
                                       <queue:target type="agent" name="'702_sip'"/>
                               </queue:targets>
                       </queue:submit>
               </onentry>

               <transition event="queue.submit.done" target="exit">
                       <log expr="'Queue Submit DONE'"/>
                       <log expr="_event.data.targetselected"/>
               </transition>

               <transition event="error.queue.submit" target="error" >
                       <log expr="'ERROR'"/>
               </transition>
       </state>

  </parallel>

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

- From the begin state, we now transition to a set of parallel states. When we enter the parallel state `routingwithdialog`, we simultaneously enter the child states `play_music` and `route_to_agent`.
- The `play_music` state is the same as before, except the duration of the music has been increased to 300 seconds (5 minutes). Once the music has been playing for 300 seconds, we will receive the `dialog.playsound.done` event, at which point we will exit the `play_music` state, as well as the `routingwithdialog` state, and enter the final state `exit`.
- The `route_to_agent` is the same as before, and will try to route the interaction to agent `702_sip`.
- If the interaction is successfully routed to agent `702_sip`, we get the `queue.submit.done` event and transition to the final state `exit`.
- If the interaction was not routed within 300 seconds, we get the `error.queue.submit` event, which triggers a transition to final state `error`.

This SCXML file will work well as long as agent `702_sip` becomes available within 300 seconds (5 minutes). Of course, we can modify this value and wait longer than 5 minutes, but what happens if agent `702_sip` never becomes available? If there are other agents, we may want to expand our agent selection to include those. A better approach is to first try to route to a particular agent, if unsuccessful, try to route to an agent group, if unsuccessful, try to route to a place, if unsuccessful, try to route to a place group, and if all those options could not successfully route the call, then give up. It would also be nice to let the caller know what the estimated wait time is. Here is what the SCXML file will look like:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <datamodel>
        <data id="reqid" expr="''" />
        <data id="ixnid" expr="''" />
  </datamodel>

  <state id="initial">
    <transition event="interaction.added" target="routingwithdialog">
               <script>
                       _data.ixnid = _event.data.interactionid;
               </script>
        </transition>
  </state>

  <parallel id="routingwithdialog">

    <state id="dialog" initial="play_estimated_wait_time">
		<state id="play_estimated_wait_time">
			<onentry>
				<dialog:play language="'English(US)'">
					<dialog:prompts type="ann">
						 <dialog:prompt interrupt="true" intid="1"/>
					</dialog:prompts>
				  </dialog:play>
			</onentry>
			<transition event="dialog.play.done" target="play_music"/>
			<transition event="error.dialog.play" target="error"/>
		</state>
		<state id="play_music">
			<onentry>
				<dialog:playsound type="'music'" resource="'music/on_hold'"
duration="'60'"/>
			</onentry>
			<transition event="dialog.playsound.done"
target="play_estimated_wait_time"/>
			<transition event="error.dialog.playsound" target="error"/>
		</state>
	</state>
	<state id="routing" initial="route_to_agent">

		<state id="route_to_agent">
			<onentry>
				<queue:submit requestid="_data.reqid"
interactionid="_data.ixnid" priority="5" timeout="60">
					<queue:targets>
						<queue:target type="agent" name="'702_sip'"/>
					</queue:targets>
				</queue:submit>
			</onentry>
			<transition event="error.queue.submit" target="route_to_agent_group">
				<log expr="'Queue Submit to Agent Group'"/>
			</transition>
		</state>

		<state id="route_to_agent_group">
			<onentry>
				<queue:submit requestid="_data.reqid"
interactionid="_data.ixnid" priority="5" timeout="60">
					<queue:targets>
						<queue:target type="agentgroup"
name="'SipGr_2'"/>
					</queue:targets>
				</queue:submit>
			</onentry>
			<transition event="error.queue.submit" target="route_to_place">
				<log expr="'Queue Submit to Place'"/>
			</transition>
		</state>

		<state id="route_to_place">
			<onentry>
				<queue:submit requestid="_data.reqid"
interactionid="_data.ixnid" priority="5" timeout="60">
					<queue:targets>
						<queue:target type="place" name="'702'"/>
```

```
                                        </queue:targets>
                                </queue:submit>
                        </onentry>
                        <transition event="error.queue.submit" target="route_to_place_group">
                                <log expr="'Queue Submit to Place Group'"/>
                        </transition>
                </state>

                <state id="route_to_place_group">
                        <onentry>
                                <queue:submit requestid="_data.reqid"
interactionid="_data.ixnid" priority="5" timeout="60">
                                        <queue:targets>
                                                <queue:target type="placegroup"
name="'SIP_PlGr2'"/>
                                        </queue:targets>
                                </queue:submit>
                        </onentry>
                        <transition event="error.queue.submit" target="error">
                                <log expr="'ERROR'"/>
                        </transition>
                </state>

                <transition event="queue.submit.done" target="exit">
                        <log expr="'Queue Submit DONE'"/>
                        <log expr="_event.data.targetselected"/>
                </transition>
        </state>

  </parallel>

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

- This time, we enter the two child states `dialog` and `routing` simultaneously as soon as we enter the `routingwithdialog` parallel state.
- The `dialog` state now has two child states, `play_estimated_wait_time` and `play_music`. As soon as the `dialog` state is entered, the `play_estimated_wait_time` state becomes the active state because it has been declared as the `initial` state.
- The `play_estimated_wait_time` will play a prompt announcing the estimated wait time before the call will get routed to an agent. When the announcement is finished, we will get the `dialog.play.done` event to trigger a transition to the `play_music` state.
- The `play_music` state is the same as before and will play music for 60 seconds, then fire the `dialog.playsound.done` event, which will trigger a transition to the `play_estimated_wait_time` state.
- The `routing` state has four child states, all trying to route the call to an agent. As soon as the `routing` state is entered, the `route_to_agent` state becomes the active state. While the state machine is in any of the four child states, the `queue.submit.done` event could be fired. Since this event has no matches in the currently active child state, it will look at the parent state `routing` and look for a transition with the event name `queue.submit.done`. This will cause a transition to the final state `exit`.
- The `route_to_agent` state will try to route the call to agent `702_sip` for 60 seconds before it fires the `error.queue.submit` event which will trigger a transition to the `route_to_agent_group` state.
- The `route_to_agent_group` state will try to route the call to agent group `SipGr_2` for 60 seconds before it fires the `error.queue.submit` event which will trigger a transition to the `route_to_place` state.
- The `route_to_place` state will try to route the call to place `702` for 60 seconds before it fires the

error.queue.submit event which will trigger a transition to the route_to_place_group state.

- The route_to_place_group state will try to route the call to place group SIP_PlGr2 for 60 seconds before it fires the error.queue.submit event which will trigger a transition to the final state error.

Next, we have a situation where we are trying to detect whether the call was created from a consult call. The following SCXML file was configured on a Routing Point, and was triggered when a primary call initiated a consult call to the Routing Point:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
                        xmlns:queue="www.genesyslab.com/modules/queue"
                        xmlns:dialog="www.genesyslab.com/modules/dialog"
                        xmlns:ixn="http://www.genesyslab.com/modules/interaction"
                        initial="global">
        <script>
                var reqid;
                var consult_ixn_id;
                var primary_ixn_id;
                var effective_ixn_id;
                var sessionStarted = false;
        </script>
        <!--*********************************************************************-->
        <state id="global" initial="initial">
                <!--*********************************************************************-->
                <state id="initial">
                        <!--This ensures the session terminates after 10 minutes-->
                        <onentry>
                                <send event="'toExit'" delay="'600s'" />
                        </onentry>
                        <transition event="interaction.added" cond="sessionStarted == false" >
                                <script>
                                        /*
                                        To avoid catching another 'interaction.added' event
(caused by 'attach') in the same state again,
                                        set sessionStarted to true. 'Attach' action could be
done in a separate state, but for the sake of
                                        simplicity and to minimize number of states it is
done here in initial state...
                                        */
                                        sessionStarted = true;
                                        /* Assign interaction IDs that will be needed later
on ... */
                                        if(
_genesys.ixn.interactions[_event.data.interactionid].voice.type == 'consult' )
                                        {
                                                consult_ixn_id = _event.data.interactionid;
                                                primary_ixn_id =
_genesys.ixn.interactions[consult_ixn_id].parentid;
                                                effective_ixn_id = consult_ixn_id;
                                        }
                                        else
                                        {
                                                consult_ixn_id = undefined;
                                                primary_ixn_id = _event.data.interactionid;
                                                effective_ixn_id = primary_ixn_id;
                                        }
                                </script>
                                <log expr="'CONSULT_EXAMPLE: consult_ixn_id = ' +
consult_ixn_id" />
                                <log expr="'CONSULT_EXAMPLE: primary_ixn_id = ' +
primary_ixn_id" />
                                <log expr="'CONSULT_EXAMPLE: effective_ixn_id = ' +
effective_ixn_id" />
                                <if cond="consult_ixn_id != undefined">
                                        <log expr="'CONSULT_EXAMPLE: Consult call started
strategy. Attaching primary call...'"/>
                                        <ixn:attach requestid="reqid"
interactionid="primary_ixn_id" />
                                <else/>
                                        <log expr="'CONSULT_EXAMPLE: Normal call started
strategy. Proceeding with session ...'"/>
                                        <send event="'toProceed'" />
                                </if>
                        </transition>
                        <transition event="interaction.attach.done"
cond="_event.data.requestid == reqid" target="prewaiting_state" />
                        <!-- error.interaction.attach event (if happened) will be caught in
global state -->
                        <transition event="toProceed" target="CUSTOM_WORKING_STATE" />
                </state>
                <!--*********************************************************************-->
                <state id="prewaiting_state">
                        <onentry>
                                <!-- This illustrates the case when the session is started by
a consult call (and that
                                call is still alive here), sometimes it makes sense to wait
for some short amount of time.
                                This time could depend on how fast TServer completes
transfer, or
                                could be done to avoid routing consult call during mute
transfer, etc. -->
                                <log expr="'CONSULT_EXAMPLE: Continuing session with some
short delay...'"/>
                                <send event="'toProceed'" delay="'1s'"
/>
                        </onentry>
                        <transition event="toProceed" target="CUSTOM_WORKING_STATE"
/>
                </state>
                <!--*********************************************************************-->
                <!--************  This is where your main logic goes ********************-->
                <!--*********************************************************************-->
                <state id="CUSTOM_WORKING_STATE" initial="route_to_agent">

                        <!-- This will try to route the call to agent 703_sip.
                         If it is not successful within 3 seconds, it will transition to
state "dialog" and play music.
                         The attribute "clearontimeout" is set to false so router will
continue trying to route to the
                         agent while the music is playing. -->
                        <state id="route_to_agent">
                                <onentry>
                                        <queue:submit requestid="reqid"
interactionid="effective_ixn_id" priority="5" timeout="3" clearontimeout="false" >
                                                <queue:targets>
                                                        <queue:target type="agent"
name="'703_sip'"/>
                                                </queue:targets>
                                        </queue:submit>
                                </onentry>
                                <transition event="error.queue.submit" target="dialog" >
                                        <log expr="'ERROR WITH QUEUE SUBMIT: ' + uneval(
_event )"/>
                                </transition>
                        </state>
                        <!-- This plays music for 60 seconds. -->
                        <state id="dialog" >
                                <onentry>
                                        <dialog:playsound requestid="reqid"
interactionid="effective_ixn_id" type="'music'" resource="'music/on_hold'" duration="60" />
                                </onentry>
                                <transition event="dialog.playsound.done.timeout" />
                                <transition event="dialog.playsound.done" target="exit"/>
                                <transition event="error.dialog.playsound" target="error">
                                        <log expr="'ERROR PLAYING MUSIC: ' + uneval(_event)"
/>
                                </transition>
                        </state>

                        <transition event="queue.submit.done" target="exit">
                                <log expr="'QUEUE SUBMIT DONE.  Ending Session.'"/>
                        </transition>
                        <transition event="interaction.partystatechanged"
cond="effective_ixn_id == _event.data.interactionid">
                                <log expr="'CONSULT_EXAMPLE: Got partystatechanged event: ' +
uneval(_event.data)" />
                        </transition>
                </state>

                <!--**********************************************************************-->
                <!--**********************************************************************-->
                <!--**********************************************************************-->
                <transition event="interaction.onmerge" cond="_event.data.frominteractionid
== consult_ixn_id && _event.data.tointeractionid == primary_ixn_id" >
                        <script>
                                consult_ixn_id = undefined;
                                effective_ixn_id = primary_ixn_id;
                        </script>
                        <log expr="'CONSULT_EXAMPLE: Effective call ID changed because of
transfer completion: ' + uneval(_event)"/>
                        <log expr="'CONSULT_EXAMPLE: consult_ixn_id = ' + consult_ixn_id" />
                        <log expr="'CONSULT_EXAMPLE: primary_ixn_id = ' + primary_ixn_id" />
                        <log expr="'CONSULT_EXAMPLE: effective_ixn_id = ' + effective_ixn_id"
/>
                </transition>
                <transition event="interaction.deleted" cond="_event.data.interactionid ==
effective_ixn_id" target="exit" >
                        <log expr="'CONSULT_EXAMPLE: Effective call is dead. Exiting...: ' +
uneval(_event)"/>
                </transition>
                <transition event="interaction.deleted" cond="_event.data.interactionid ==
primary_ixn_id && consult_ixn_id != undefined" target="exit" >
                        <log expr="'CONSULT_EXAMPLE: Primary call is dead, consult call is
alive and useless. Exiting...: ' + uneval(_event)"/>
                </transition>
                <!--In case none of the other events are triggered, this will end the session
after number of minutes specified at the strategy beginning-->
                <transition event="toExit" target="exit">
                        <log expr="'CONSULT_EXAMPLE: Possibly stuck session is self-
destructing. Exiting...: ' + uneval(_event)"/>
                </transition>
                <!--This will catch all the errors that are not processed elsewhere-->
                <transition event="error.*" target="error" >
                        <log expr="'CONSULT_EXAMPLE: ERROR AT GLOBAL LEVEL'"/>
                        <log expr="'CONSULT_EXAMPLE: Got error event: ' + uneval( _event )" />
                </transition>
        </state>
        <final id="exit"/>
        <final id="error"/>
</scxml>
```

- When an agent that is part of the primary call initiates a transfer or consult to the Routing Point, it will trigger a SCXML session to be created and will wait for the `interaction.`added event.
- After the `interaction.`added event is received, it will set the `consult_ixn_id`, `primary_ixn_id`, and `effective_ixn_id` depending on whether the session was started by a regular call, or a consult call to the Route Point.
- If the call that started the session is a `consult` call, we attach the parent interaction (the primary call which is ownerless) to the current session (see interaction attach for more details about ownership).
- The `interaction.attach.done` event will trigger a transition to the `prewaiting_state`, where we put in a delay. This delay is needed depending on how fast TServer completes the transfer, or is sometimes done to avoid routing a consult call during a mute transfer.
- The `CUSTOM_WORKING_STATE` is where you would put your main logic. In this example, we first try to route the call to agent `703_sip`. If this is not successful within 3 seconds, we transition to the `dialog` state and play music for 60 seconds.
- At any time during the session, if the transfer or consult is completed, the `interaction.onmerge` event will be triggered and various interaction IDs will be updated. This is needed to because the consult call is deleted during the merge. The `consult_ixn_id` will no longer be valid and is set to `undefined`. The `effective_ixn_id` is updated and should be used from this point forward for all functions and actions that require an interaction ID.
- Exiting the session is triggered by any of the following situations:
    - The call is successfully routed to agent `703_sip`.
    - Music has been played for 60 seconds.
    - There was a problem playing the file `music/on_hold`.
    - The effective call is deleted (effective call is the consult call until the consult or transfer is complete, at which time, it is the only call left).
    - The primary call is deleted before the consult or transfer is complete (the consult call can still be alive but is useless at this point).
    - Any `error.*` events that are raised during the session.
    - The session may be stuck and self-destucts 10 minutes after it was created.

Orchestration Server How-To

Timers and Wait Functions

In SCXML, orchestration logic can implement timers and wait functions using the `<send>` element and the delay attribute. A state can send a delayed event to itself and then when the time expires the event will be processed by the `<transition>` element defined in the state to process it.

Modularity

There are several ways to support modularity and reusability with SCXML:

- xinclude <xi:include /> - This is an XML standard for including other documents into another XML document, by providing a macro-like functionality. Note: For details on how the orchestration platform will support this, see **<xi:include>**.
- <invoke> - This creates a stand-alone sub-state machine that communicates asynchronously with its parent. See SCXML <invoke> for details.
- Targetless transitions within a state can provide subroutine-like functionality for the state and all its contained states. What is needed are two events:
    - The input event - This defines the name of the subroutine and the input parameters to the subroutine.
    - The output event - This defines the event that the invoking state should wait for to get the results of the subroutine.

The subroutine is implemented as follows:

-
    - Either in `<onentry>` or within one or more child `<states>`, generate the input event. This can be done using `<raise>/<event>`.
    - Define a targetless transition in the parent state that will handle the input event. This will provide the body of the subroutine. The body of the subroutine should generate the output event. This can be done using <raise>/<event> or by calling an action that will generate the output event.
    - Define a transition that will handle the output event. This transition can access the information in the output event to determine the results of the subroutine. This transition can be targetless. If targetless, it can act as another step in a more complex subroutine. If it is not targetless, it should be defined at the same level where the subroutine's input event was generated.

The following is an example of a targetless transition style subroutine. The input event is "inputsub1" and the output event is "outputsub1". Several steps are chained together to provide a moderately complex subroutine. This mechanism should only be used with events generated by the steps of the subroutine; otherwise asynchronous events generated by actions could result in steps not being executed properly.

```
<state id="ParentState">
        <transition event="inputsub1">
                <script>
                        local.eMailID =
_genesys.getValue(_event.data.i_ixn,"'InteractionId'");
                </script>
                <log expr="_event.data.i_message" level="3"/>
                <session:fetch requestid="_data.reqid" srcexpr="'someURL/AUDIT_PROC'"
timeout="10">
                        <param name="audit_info" expr="_event.data.i_message"/>
                        <param name="message_id" expr="local.eMailID"/>
                </session:fetch>
        </transition>
                <!-- This an example of branching within a targetless transition subroutine --
>
        <!-- It examines the event generated by the session:fetch action -->
        <!-- If value1 is less than or equal to 10 go to step 2. -->
        <transition event="session.fetch.done" cond="(_event.requestid == _data.reqid) &&
(_event.data.content.value1 <= "10")">
                <raise event="sub1step2">
                        < param name ="s1v1" expr ="_event.data.value1"/>
                </raise>
        </transition>
                <!-- If value1 is greater than to 10 go to step 3. -->
        <transition event="session.fetch.done" cond="(_event.requestid == _data.reqid) &&
(_event.data.content.value1 > 10)">
                <raise event="sub1step3">
                        < param name ="s1v1" expr ="_event.data.value1"/>
                </raise >
        </transition>
        <!-- This is the processing for step 2 of the subroutine sub1 -->
        <transition event="sub1step2" >
                <script>
                        <! - do some extra processing -->
                </script>
                <!-- Return to the involving state -->
                <raise event="outputsub1">
                        <param name ="sub1op1" expr ="variablex"/>
                        <param name ="rc" expr ="success"/>
                </raise>
        </transition>
        <!-- This is the processing for step 3 of the subroutine sub1 -->
        <transition event="sub1step3" >
                <script>
                        <!-- do some extra processing -->
                </script>
                <if conn="variable >=100">
                        <!-- Return to the involving state -->
                        <raise event="outputsub1">
                                <param name ="sub1op1" expr ="variablex"/>
                                <param name ="rc" expr ="success"/>
                        </raise>
                </else >
                        <!-- go to step 4 -->
                        <raise event="sub1step4">
                                <param name ="sub1op1" expr ="variablex"/>
                                <param name ="sub1op1" expr ="variabley"/>
                        </raise>
                </if>
        </transition>
        <!-- This is the processing for step 4 of the subroutine sub1 -->
        <transition event="sub1step4" >
                <script>
                        <!-- do some extra processing -->
                </script>
                <!-- Return to the involving state -->
                <raise event="outputsub1">
                        <param name ="sub1op1" expr ="variablex"/>
                        <param name ="rc" expr ="success"/>
                </raise>
        </transition>
        <!-- General error processing for this sub1 -->
        <!-- Note that this will treat all error events as a failure of this subroutine -->
        <!-- Care should be taken to ensure that this is only called as part of the
subroutine -->
        <transition event="error.*" cond="_event.requestid == _data.reqid">
                <log expr="had an error with the fetch" level="3"/>
                <raise event="outputsub1">
                        < param name ="rc" expr ="fetchfailed"/>
                </raise >
        </transition>
        <!-- This is the state that invokes the subroutine -->
        <state id="stepwhichinvokessub1">
                <onentry>
                        <raise name ="inputsub1">
                                <param name ="i_message" expr ="'here is the message'">
                                <param name ="i_ixn" expr ="_data.interaction"">
                        </raise">
                </onentry>
                <transition event="outputsub1">
                        <if cond = (_event.data.rc == "success")>
                                <!-- do the processing to continue based on sub1 completing --
>
                        </else >
                                <!-- do the processing to continue based on sub1 failing -->
                        </if>
                </transition>
        </state>
</state>
```

Using <invoke>

In addition to what the SCXML specification defines, the following guidelines can be followed when developing an asynchronous subroutine with this functionality:

- Invoked document/session:
  - Should use `<datamodel>` and `<data>` for mapping of the invoking session's `<param>`s to the invoked session's data model. This is the primary means of passing data to the invoked session.
  - Must use the `<donedata>` element as a child of the top-level `<final>` state(s). This will allow the invoked state machine to return the appropriate output parameters in the done.invoke event. This event is sent to the invoking session.
  - Must not send explicit events to the including document/session.
  - Should not rely on events from the including document/session, however, the subroutine can use cancel.invoke to perform any cleanup necessary if the subroutine is interrupted. This can happen if the parent session transitions out of the invoking state.
- Invoking document/session:
  - Should use `<param>` to provide argument information to the invoked session. Since the sessions do not share a `<datamodel>`, this is the primary means for passing data to the invoked session.
  - Must have a `<transition>` for the done.invoke event that will be generated by the invoked session. This allows the invoked session to communicate the subroutine results back to the invoking document/session and transition to the next state based on the results.

Handling Assembly and Compilation Problems

When using xinclude, developers should ensure that valid documents and fragments are used. This will avoid many XML parsing problems. However, since large SCXML documents can be very complicated, it is still possible to have document errors that prevent the final application from being assembled, parsed, and compiled. When dealing with such a situation, a developer can place the following in the main application document:

```
<!-- $$_GENESYS_DEBUGGING_$$ -->
```

If the application document cannot be parsed and compiled, and this element is present in the document, the entire application document will be written to a file in the current working directory. The filename will be sessionid.scxml, where sessionid is the ID of the session that was being created. The information contained in this file should be sufficient to allow the developer to determine why the failure occurred. These files need to be removed manually by the developer after the problems have been resolved.

External Interfaces

The orchestration platform has a set of RESTFul Web 2.0 Web Services APIs. These APIs allow the following interaction with SCXML sessions:

- **Start SCXML Session** - This action starts a new orchestration application instance (session).
- **Stop SCXML Session** - This action terminates a given orchestration application instance (session).
- **Publish Event to SCXML Session** - This action allows an external system to send an event to a given orchestration session.
- **Send a Request to SCXML Session** - This action allows an external system to send a request to a given session to be processd. The session will respond to this request with the `<response>` element.
- **Query SCXML Session** - This action gets the requested orchestration session data. This can be used to get current session data on any session.

> **Tip**
> (Starting with 8.1.400.55), the `/ors/help` method provides help for ORS REST APIs.

These external interfaces may have equivalent functionality in SCXML or Orchestration extensions. The following table reflects the mapping between the two.

External RESTful APIs	SCXML or Functional Module Function
`http://<server:port>/scxml/session/start`	`<session:start>`
`http://<server:port>/scxml/session/<session id>/terminate`	`<session:terminate>`
`http://<server:port>/scxml/session/<session id>/event/<name>[?<parameters>]`	`<scxml:send>`
`http://<server:port>/scxml/session/<session id>/request/<name>[?<parameters>]`	`<session:fetch>`
`http://<server:port>/scxml/session/<session id>/query`	none

Start SCXML Session

This API starts a new SCXML session for a specific application. The "src" parameter is mandatory. All of the other parameters are optional.

`http://<server:port>/scxml/session/start`		
HTTP Verbs	PUT	Not used
	POST	Used to start a given session. (supported Content-Type - application/x-www-form-urlencoded)
	DELETE	Not used
	GET	Not used
URI-Variable Elements	none	
Request-URI Parameters	none	

Document Body - using
application/x-www-form-urlencoded
	src	URL of the SCXML document to use for the new session.
		Starting with 8.1.400.09, Orchestration Server provides the ability to use the Enhanced Routing Script object when starting a new session by web request. The script name in the `script:ScriptName` format can be defined as a value of the `src` parameter of the `/scxml/session/start` web request. Example:
		`http://localhost:7031/scxml/session/start?src=script:Script1`
		Note: If you are using an Enhanced Routing Script object that exists under a Tenant other than Environment, the Tenant must be explicitly specified as a URL attribute. For example:
		`http://<host>:<port>/scxml/session/start?src=script:<scriptname>&tenant=<Tenant`

	idealtime	A dateTime value which will represent the date and time that this session is to be started. This value should be the time as returned by the ECMAScript Date(...).getTime() function, which is given in the number of milliseconds since 00:00:00 UTC on January 1, 1970.
	request-specific	These are request-specific parameters. These parameters will be put in the

`http://<server:port>/scxml/session/start`		
		appropriate session data items (`<data>`) when the session is initiated if the name of the parameter matches the ID attribute of the `<data>` element. For example, if you have the following parameters, `p1=12355, p2=abcd`, then `<data id="p1>` and `<data id="p2">` will be set to the corresponding values. If a parameter value does not match the ID of a data item, it will be thrown away. In addition, there will be a parameter which specifies the content type for the parameters and it will be set to "application/x-www-form-urlencoded".
	results	A body parameter value which will represent a callback URL that accepts the results of a scheduled session start, whether it has positive or negative results. This URL will be invoked with the HTTP POST method. The content of the document body will be the following: • Type - This is the type of response - positive or negative. • Reason - This is the reason why the response was generated. • sessionid - This is the ID of the session that is being started. • server - This is the URL of the server that can be used to invoke other requests on the same session. (Question: Should this be merged with the sessionid?)
	sid	ID of the session that will be created. If this parameter is missing, the session will have a standard ID that will be generated by ORS.
	prewindow	Duration in milliseconds representing the time window prior to the ideal time in which the session could be started.
	postwindow	Duration in milliseconds representing the time window after the ideal time in which the

http://<server:port>/scxml/session/start		
		session could be started.
Positive Response (200 Response)	ID	The identifier of the newly created session.
Negative Response	HTTP Error Code	HTTP 4xx

Example	`POST http://<server:port>/scxml/session/start`
	`. . .`
	`Content-type: application/x-www-form-urlencoded`
	`. . .`
	`src=http://appserver/appname.scxml`	

Stop SCXML Session

This API terminates a SCXML session.

http://<server:port>/scxml/session/<session id>/terminate		
HTTP Verbs	PUT	Not used
	POST	Stops an existing session.
	DELETE	Not used
	GET	Not used
URI-Variable Elements	session id	This identifies the session which is to be stopped.
Document Body	none	none
Positive Response (200 Response)	OK	OK
Negative Response	HTTP error code	HTTP 4xx
Example	`POST http://<server:port>/scxml/session/1234567/terminate`	

Query SCXML Session

This API queries a given SCXML session's data.

http://<server:port>/scxml/session/<session ids>/query		
HTTP Verbs	PUT	Not used
	POST	Not used
	DELETE	Not used
	GET	Query a set of existing sessions.
URI-Variable Elements	session ids	This identifies the set of sessions which are to be queried. The session ids are separated by a "," For example, `[http://<server:port>/scxml/session/123456,345677,66778898]`. Currently, we only support a list of one session id.
Request-URI Parameters	none	
Document Body	none	none
Positive Response (200 Response)	sessionData	This is a list of sessions and their associated data. The following is the JSON-formatted set of session data which will be returned for each session in the list: • Session ID • URL of the application • Name - _name attribute • Type - _type attribute • Current states • Current events • _data properties (application-related data) • _genesys properties (functional module-related data)
Negative Response	HTTP error code	404 Not found
Example	`GET http://<server:port>/scxml/session/1234567/query`	

Send Request to SCXML Session

This API sends a request to the SCXML session to process.

`http://<server:port>/scxml/session/<session id>/request/<name>`		
HTTP Verbs	PUT	Not used
	POST	Send a request to an existing session.
	DELETE	Not used
	GET	Not used
URI-Variable Elements	session id	This is the session which the request is targeted for.
	name	This is the name of the request which is to be performed by the SCXML session when it receives the corresponding request event. This value will be the name of the SCXML event which the SCXML session will process.
Request-URI Parameters	none	
Document Body - can be in any of the following encodings • application/x-www-form-urlencoded • application/json • text/xml	request-specific	These are request-specific parameters. These parameters will be put into the SCXML event at the following location: _event.data.param.xxx. For example, if you have the following body parameters, p1=12355, p2=abcd, then the following will be the structure in the event: _event.data.param.p1 and _event.data.param.p2. In addition, there will be a parameter which specifies the content type for the parameters and it will be set based HTTP Content-Type element value. For details see the [[|API Parameter passing]] section.
	request identifier	This API-related parameter is generated by the orchestration platform when it gets the HTTP request. It is used to correlate the requests and the corresponding responses for a given session. This identifier is put in the sendid property of the request's SCXML event (that is, _event.sendid) when sent to the application. This identifier must be used in the corresponding `<response>` element.
Positive Response (200 Response)	results	This is a set of data, based on the results of the request. It is request-specific in its content. The SCXML session sends this data via the [[|<response>]] element.
	headers	This is a collection of key-value

http://<server:port>/scxml/session/<session id>/request/<name>		
		pairs, representing a select group of request headers obtained from the HTTP request. It is request-specific in its content. This data can be accessed in the SCXML event at the following location: `_event.data.headers` The list of retrievable headers are as follows: • HTTP_METHOD, HTTP_VERSION, HTTP_REQUEST_URI, ACCEPT, DATE, USER-AGENT, CONNECTION, ACCEPT-LANGUAGE, REFERER, IF-MODIFIED-SINCE, FROM, MIME-VERSION, PRAGMA, AUTHORIZATION, CONTENT-LENGTH, CONTENT-TYPE, CONTENT-ENCODING
Negative Response	HTTP error code	HTTP 4xx

Example	`POST http://<server:port>/scxml/session/1234567/request/`
	`getxData`
	`. . .`
	`Content-type: application/x-www-form-urlencoded`
	`. . .`
	`parm1=john`
	
	The SCXML session must have the following SCXML snippet somewhere in its definition. This type of request processing SCXML snippet is probably best placed as a global document event handler.
	
	`<transition event="getxData" cond="_event.data.param.parm1`
	`= '' " >`
	`        <ws:response requestid="_event.sendid"`
	`type="negative"`
	`                resultcode="invalidparameter"/>`
	`</transition>`
	`<transition event="getxData"`
	`cond="_event.data.param.parm1 != '' " >`
	`        <script>`
	`                var rdata =`
	`_getdata(_event.data.param.parm1);`
	`        </script>`
	`        <ws:response requestid="_event.sendid" >`
	`                <param name="results" expr="rdata"/>`
	`        </ws:response>`
	`</transition>`	

Special Considerations	It is recommended that the orchestration logic for processing this event should be processed within the executable content of the `<transition>` element that receives the request event. This includes	

`http://<server:port>/scxml/session/<session id>/request/<name>`	
	using the `<response>` element to respond to the request. If the processing of the request requires more complicated processing (for example, it must transition to a sub-state model for processing), then the application must copy all the necessary request data (for example, _event.sendid) from the event and put it into the appropriate global variables so that the sub-state model can use it to process the request. This is because the event data becomes invalid after the transition element has been processed.

Publish Event to SCXML Session

This API sends an event to an existing session.

`http://<server:port>/scxml/session/<session id>/event/<name>`		
HTTP Verbs	PUT	Not used
	POST	Send an event to an existing session.
	DELETE	Not used
	GET	Not used
URI-Variable Elements	session id	This is the session which the event is targeted for.
	name	This is the name of the event which is to be sent to the SCXML session. This value will be the name of the SCXML event which the SCXML session will process.
Request-URI Parameters	none	
Document Body - can be in any of the following encodings • application/x-www-form-urlencoded • application/json • text/xml	request-specific	These are event-specific parameters. These parameters will be put into the SCXML event at the following location: _event.data.param.xxx. For example, if you have the following body parameters, p1=12355, p2=abcd, then the following will be the structure in the event: _event.data.param.p1 and _event.data.param.p2. In addition, there will be a parameter which specifies the content type for the parameters and it will be set based HTTP Content-Type element value. For details see API Parameter Passing.
Positive Response (200 Response)	none	**Note:** (Since 8.1.200.45) Behaviour depends on configuration option

http://<server:port>/scxml/session/<session id>/event/<name>		
		[Orchestration/ webfm_event_hold_response] (default: true) When [Orchestration/ webfm_event_hold_response] is true, no response is given until the event is processed. The response status code depends on how the event is processed: • Event is processed and transition is taken - OK 200 Response • Event is processed and no transition is taken - No Content 204 Response When [Orchestration/ webfm_event_hold_response] is false, 200 Response is provided immediately. Prior to 8.1.200.45, 200 Response is provided immediately.
	headers	This is a collection of key-value pairs, representing a select group of request headers obtained from the HTTP request. It is request-specific in its content. This data can be accessed in the SCXML event at the following location: _event.data.headers The list of retrievable headers are as follows: • HTTP_METHOD, HTTP_VERSION, HTTP_REQUEST_URI, ACCEPT, DATE, USER-AGENT, CONNECTION, ACCEPT-LANGUAGE, REFERER, IF-MODIFIED-SINCE, FROM, MIME-VERSION, PRAGMA, AUTHORIZATION, CONTENT-LENGTH, CONTENT-TYPE, CONTENT-ENCODING
Negative Response	HTTP error code	HTTP 4xx
Example	`POST http://<server:port>/scxml/session/1234567/event/xisDone` `. . .` `Content-type: application/x-www-form-urlencoded` `. . .`	

http://<server:port>/scxml/session/<session id>/event/<name>	

	`parm1=john`
	
	The SCXML session must have the following SCXML snippet somewhere in its definition. This type of request-processing SCXML snippet is probably best placed as a global document event handler.
	
	`<transition event="xisDone" target="continuewithY" >`
	`        <! - Do some processing here ..../>`
	`</transition>`

Orchestration Platform Status

The status of the orchestration platform can be obtained simply by using a GET HTTP request. The following are the URLs for different types of platform status. The results from the request are in XML.

URL	Type	Results
`<Orch platform server>:<port>`	Basic data	• Version • Start time • Running time
`<Orch platform server>:<port>/server?cfgStatistics`	Configuration data	
`<Orch platform server>:<port>/server?activeCalls`	Interaction data	
`<Orch platform server>:<port>/server?scxmlStat`	SCXML engine data	
`<Orch platform server>:<port>/serverx?activeApplications`	SCXML Application data	For each active application, the following information is provided: • URL • Name • startedSessions • endedSessions • abortedSessions

API Parameter Passing

Parameters are passed from the Web 2.0 APIs to the SCXML session using two basic mechanisms. These mechanisms are **mutually exclusive**:

URL-Encoded Parameters

URL-encoded parameters can be specified in a request. Parameters may be specified either in the URL for GET based APIs:

```
GET http://<server:port>/scxml/session/<session id>/event/<name>?param1=1¶m2=2
```

Or in the document body for POST and PUT based APIs:

```
POST http://<server:port>/scxml/session/<session id>/event/<name>
.. . .
Content-type: application/x-www-form-urlencoded
. . .
param1=1¶m2=2
```

In both cases, URL encoded parameters will be translated to properties of the SCXML event:

```
_event.data.param.param1 = 1
_event.data.param.param2 = 2
```

The _event.data.paramtype will be set to application/x-www-form-urlencoded.

In the case of the `[http:// http://]<server:port>/scxml/session/start` API (Start Session), the following processing will be done:

- If the name of the parameter matches the 'id' of a `<data>` element in the data model of the started session, then the value of the parameter will replace the value of the corresponding `<data>` element.
- If the names do not match, the value of the parameter will not be added to the started session's data model.

Document Body-Encoded Parameters

These parameters are specified in the document body using the following supported content types:

- **application/json** - In this case, the body of the request contains JSON-encoded data. The whole body of the request is passed to the SXCML session as a value of the "_event.data.param" event property. In addition, the "_event.data.paramtype" property will be set to "application/json", based on the HTTP Content-Type element. For example:

```
POST http://<server:port>/scxml/session/<session id>/event/<name>
...
Content-Type=application/json
Content-Length=xx
...
param="{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 555-1234",
        "646 555-4567"
    ]}"
```

An SCXML application may process these parameters by evaluating JSON text to ECMAScript objects. In the case of the [http:// http://]<server:port>/scxml/session/start API (Start Session), the following processing will be done:

- If the name of the parameter matches the 'id' of a `<data>` element in the data model of the started session, then the value of the parameter will replace the value of the corresponding `<data>` element.
- If the names do not match, the value of the parameter will not be added to the started session's data model.

text/xml - In this case, the body of the request contains XML-encoded data. The whole body of the request is passed to the SXCML session as a value of the "_event.data.param" event property. In addition, the "_event.data.paramtype" property will be set to "text/xml". For example:

```
POST http://<server:port>/scxml/session/<session id>/request/<name>
...
Content-Type=text/xml
Content-Length=xx
...
param="
<findCar>
    <make>Dodge</make>
    <model>Daytona</model>
</findCar>
"
```

An SCXML application may process this parameter using ECMAScript XML capabilities. In the case of the http://server:port/scxml/session/start API (Start Session), the following processing will be done:

- If the name of the parameter matches the 'id' of a `<data>` element in the data model of the started session, then the value of the parameter will replace the value of the corresponding `<data>` element.
- If the names do not match, the value of the parameter will not be added to the started session's data model.

Orchestration Server Sample Applications

Using The Queue Module

Using ECMAScript

Fetching Data

Invoking SCXML Sessions

Using The Interaction Interface

- Access Interaction Properties
- Set User Data
- Delete User Data
- Delete All User Data

Using The Voice Interaction Interface

Using The Dialog Interface

Using The Statistics Interface

- Get Statistic Value
- Get Average Statistic Value
- Get Minimum Statistic Value
- Get Maximum Statistic Value

Using The Session Interface

Using Multimedia

- Work With E-Mail Or SMS
- Work With Chat

Route to DN

The following SCXML strategy routes an interaction to a DN. When routing is completed, the strategy outputs information to the Orchestration Server (ORS) log.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetobjectselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to DN Using Target ID

The following SCXML strategy routes an interaction to a DN, using the target id format.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:targetid expr="'7102@.DN'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Handle Routing Failure

This SCXML strategy demonstrates how to handle a situation when routing fails.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="5">
        <queue:targets type="dn">
          <queue:target name="'7400'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="routing2"/>
  </state>

  <state id="routing2">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to DN and Put to Virtual Queue

The following SCXML strategy routes an interaction to a DN and puts the interaction into a virtual queue.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit queue="'VQ_10001_URS_UT_SIPSWITCH_HOME'" priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent

The following SCXML strategy routes an interaction to an agent.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:target type="agent" name="'az'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent Using Target ID

The following SCXML strategy routes an interaction to an agent, using the target id format.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:targetid expr="'az@.A'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent Without Checking Ready State

The following SCXML strategy routes an interaction to an agent with free DNs, even if Stat Server reports the agent as not ready.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing">
      <script>
        _data.ixnid = _event.data.interactionid;
      </script>
    </transition>
  </state>
  <state id="routing">
    <onentry>
      <script>_genesys.queue.checkAgentState(_data.ixnid, false);</script>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:target type="agent" name="'az'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent on Specific DN Type

The following SCXML strategy routes an interaction to an agent only if the agent has free DNs of a specified type.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing">
      <script>
        _data.ixnid = _event.data.interactionid;
      </script>
    </transition>
  </state>
  <state id="routing">
    <onentry>
      <script>_genesys.queue.useDNType(_genesys.ixn.interactions[_data.ixnid].g_uid,
_genesys.resource.resourceType.CFGACDPosition);</script>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:target type="agent" name="'az'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent and Run Treatments in Parallel

The SCXML strategy below demonstrates how to use parallel states. It applies a series of treatments while routing the interaction to an agent.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routingwithdialog"/>
  </state>
  <parallel id="routingwithdialog">
    <state id="dialog" initial="prompt">
      <state id="prompt">
        <onentry>
          <dialog:play language="'English_US'">
            <dialog:prompts type="ann">
              <dialog:prompt interrupt="true" text="'You reached Genesys'"/>
            </dialog:prompts>
          </dialog:play>
        </onentry>
        <transition event="dialog.play.done" target="music"/>
        <transition event="error.dialog.play" target="error"/>
      </state>
      <state id="music">
        <onentry>
          <dialog:playsound type="'music'" resource="'MusicDN'"/>
        </onentry>
        <transition event="error.dialog.playsound" target="error"/>
      </state>
    </state>

    <state id="routing">
      <onentry>
        <queue:submit priority="5" timeout="80">
          <queue:targets>
            <queue:target type="agent" name="'az'"/>
          </queue:targets>
        </queue:submit>
      </onentry>
      <transition event="queue.submit.done" target="exit">
        <log expr="'DONE'"/>
        <log expr="_genesys.ixn.interactions[0].voice.ani"/>
        <log expr="'DONE'"/>
        <log expr="_event.data.targetselected"/>
      </transition>
      <transition event="error.queue.submit" target="error"/>
    </state>

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

Route to Place

The following SCXML strategy routes an interaction to a place.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:target type="place" name="'pl_0002'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Place Using Target ID

The following SCXML strategy routes an interaction to a place, using the target id format.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial" >
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:targetid expr="'pl_0002@.AP'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Place Without Checking Ready State

The following SCXML strategy routes an interaction to a place that has free DNs, even if Stat Server reports it as not ready.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing">
      <script>
        _data.ixnid = _event.data.interactionid;
      </script>
    </transition>
  </state>
  <state id="routing">
    <onentry>
      <script>_genesys.queue.checkAgentState(_data.ixnid, false);</script>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:target type="place" name="'pl_0002'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent Group

The following SCXML strategy routes an interaction to an agent group.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:target type="agentgroup" name="'ag_0002'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent Group Using Target ID

The following SCXML strategy routes an interaction to an agent group, using the target id format.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:targetid expr="'ag_0002@.GA'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent Group Without Checking Ready State

The following SCXML strategy routes an interaction to an agent group that has free DNs, even if Stat Server reports it as not ready.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing">
      <script>
        _data.ixnid = _event.data.interactionid;
      </script>
    </transition>
  </state>
  <state id="routing">
    <onentry>
      <script>_genesys.queue.checkAgentState(_data.ixnid, false);</script>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:target type="agentgroup" name="'ag_0002'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent Group With Threshold

The following SCXML demonstrates the use of a statistic as a threshold for routing. It routes an interaction to an agent group only if it has exactly one available agent.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:target type="agentgroup" name="'ag_0002'" threshold= "'sdata(ag_0002.GA,
StatAgentsAvailable)=1'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Place Group

The following SCXML strategy routes an interaction to a place group.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial" >
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="placegroup">
          <queue:target name="'pg_0002'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Place Group Using Target ID

The following SCXML strategy routes an interaction to a place group, using the target id format.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial" >
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:targetid expr="'pg_0002@.GP'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Place Group Without Checking Ready State

The following SCXML strategy routes an interaction to a place group that has free DNs, even if Stat Server reports it as not ready.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing">
      <script>
        _data.ixnid = _event.data.interactionid;
      </script>
    </transition>
  </state>
  <state id="routing">
    <onentry>
      <script>_genesys.queue.checkAgentState(_data.ixnid, false);</script>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="placegroup">
          <queue:target name="'pg_0002'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Queue

The following SCXML strategy routes an interaction to a queue.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="queue">
          <queue:target name="'8002_URS_UT_SIPSWITCH_HOME'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Queue Using Target ID

The following SCXML strategy routes an interaction to a queue, using the target id format.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:targetid expr="'8002_URS_UT_SIPSWITCH_HOME@.Q'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent by Skills

The following SCXML strategy routes an interaction to an agent with a particular set of skills.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:target skillexpr="'Checking > 5 & English > 8'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent by Skills Using Target ID

The following SCXML strategy uses the target id format to route an interaction to an agent with a particular set of skills.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:targetid expr="'?:Checking > 5 & English > 8@.GA'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Routing Point

The following SCXML strategy routes an interaction to a routing point.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="routepoint">
          <queue:target name="'5002_URS_UT_SIPSWITCH_HOME'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Routing Point Using Target ID

The following SCXML strategy routes an interaction to a routing point, using the target id format.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:targetid expr="'5002_URS_UT_SIPSWITCH_HOME@.RP'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Multiple Agents

The following SCXML strategy routes an interaction to more than one agent.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="agent">
          <queue:target name="'bz'"/>
          <queue:target name="'az'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Multiple Agents Using Target ID

The following SCXML strategy routes an interaction to more than one agent using the target id format.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets>
          <queue:targetid expr="'bz@.A,az@.A'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Route to Agent Using Statistic

The following SCXML strategy routes an interaction to one of two agents based on a statistic. It requests the statistic value first, waits for this value to become available using a special state called delay, and then uses this value for routing.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:statistic="www.genesyslab.com/modules/statistic"
        initial="initial">
  <state id="initial">
<!--
    <transition event="interaction.added" target="statistics"/>
-->
    <transition event="interaction.added" target="subscribe"/>
  </state>
<!--
  <state id="routing">
    <onentry>
      <queue:submit orderstat="'StatTimeInReadyState'" ordertype="'max'" priority="5"
clearontimeout="true" timeout="0">
        <queue:targets type="agent">
          <queue:target name="'az'"/>
          <queue:target name="'bz'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="error"/>
    <transition event="error.queue.submit" target="delay"/>
  </state>
-->
<!--
  <state id="statistics">
    <onentry>
      <script>
        var t1 = _genesys.statistic.sData('az@.A', 'StatTimeInReadyState');
        var t2 = _genesys.statistic.sData('bz@.A', 'StatTimeInReadyState');
      </script>
    </onentry>

    <transition target="delay"/>
  </state>

  <state id="delay">
    <onentry>
      <send event="'delay'" target="_sessionid" targettype="'scxml'" delay="'2s'"/>
    </onentry>
    <transition event="delay" target="routing"/>
  </state>
-->
  <state id="subscribe">
    <onentry>
      <statistic:subscribe object="'az@.A'" statistic="'StatTimeInReadyState'" interval="0"/>
      <statistic:subscribe object="'bz@.A'" statistic="'StatTimeInReadyState'" interval="0"/>
    </onentry>
    <transition event="statistic.subscribe.done" target="delay"/>
    <transition event="error.statistic.subscribe" target="error"/>
  </state>
  <state id="delay">
    <onentry>
      <send event="'delay'" target="_sessionid" targettype="'scxml'" delay="'2s'"/>
    </onentry>
    <transition event="delay" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit orderstat="'StatTimeInReadyState'" ordertype="'max'" priority="5"
timeout="20">
        <queue:targets type="agent">
          <queue:target name="'az'"/>
          <queue:target name="'bz'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Set Default Destination

The sample below sets the default destination for calls.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing">
      <script>
        _data.ixnid = _event.data.interactionid;
      </script>
    </transition>
  </state>
  <state id="routing">
    <onentry>
      <script>_genesys.session.setOptions(_data.ixnid,
        'default_destination', '7102');</script>
      <queue:default/>
    </onentry>

    <transition event="queue.default.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.default" target="error"/>
  </state>

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

ECMA Script on Top Level

The following SCXML strategy uses an ECMA script fragment on the top level.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <script>
    var DN = "";
    var DNPreffix = "710";
    for (var i = 0; i < 3; i++)
      DN = DNPreffix + i;
  </script>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>
    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>
  <final id="exit"/>
  <final id="error"/>
</scxml>
```

ECMA Script on Entry Into State

The following SCXML strategy uses an ECMA script fragment on entry into a state.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <script>
        var DN = "";
        var DNPreffix = "710";
        for (var i = 0; i < 3; i++)
          DN = DNPreffix + i;
      </script>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

ECMA Script on Exit From State

The following SCXML strategy uses an ECMA script fragment on exiting from a state.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <script>
    var DN = "";
    var DNPreffix = "710";
  </script>
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
    <onexit>
      <script>
        for (var i = 0; i < 3; i++)
          DN = DNPreffix + i;
      </script>
    </onexit>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

ECMA Script During Transition

The following SCXML strategy uses an ECMA script fragment in transitioning from one state to another.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <script>
    var DN = "";
    var DNPreffix = "710";
  </script>
  <state id="initial">
    <transition event="interaction.added" target="routing">
      <script>
        for (var i = 0; i < 3; i++)
          DN = DNPreffix + i;
      </script>
    </transition>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

ECMA Script After Invoke

The following SCXML strategy uses an ECMA script fragment in the <finalize> clause after invoking another strategy.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="invocation"/>
  </state>
  <script>
    var DN = "";
    var DNPreffix = "710";
  </script>
  <state id="invocation">
    <invoke src="'http://localhost:9090/strategies/01_BASIC/_aux/DoNothing.xml'"
        type="scxml">
      <finalize>
        <script>
          for (var i = 0; i < 3; i++)
            DN = DNPreffix + i;
        </script>
      </finalize>
    </invoke>
    <transition event="done.invoke.invocation.*" target="routing"/>
    <transition event="error.invoke.invocation.*" target="error"/>
  </state>

  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

ECMA Script Function

The following SCXML strategy illustrates an example of an ECMA script function.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <script>
    function getDN()
    {
      var DN = "";
      var DNPreffix = "710";
      for (var i = 0; i < 3; i++)
        DN = DNPreffix + i;
      return DN;
    }
  </script>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="getDN()"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

ECMA Script and Data Model

The following SCXML strategy uses an ECMA script to access the strategy's data model.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="routing"/>
  </state>
  <datamodel>
    <data ID="DN"/>
  </datamodel>
  <script>_data.DN="7102";</script>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="_data.DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Fetch Data

The following sample shows how to fetch a .jsp resource from a Web Server. The fetched resource returns a DN that can be used for routing purposes. The logic of calculating the target DN is implemented on the Web Server side and it is the .jsp page creator who determines how the DN is calculated.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:session="www.genesyslab.com/modules/session"
        initial="initial">
  <script>var DN = "";</script>
  <state id="initial">
    <onentry>
      <script>var URI="http://localhost:9090/strategies/01_BASIC/_aux/GetDN.jsp";</script>
      <session:fetch srcexpr="URI"/>
    </onentry>
    <transition event="session.fetch.done" target="check">
      <script>DN = _event.data.content;</script>
    </transition>
    <transition event="error.session.fetch" target="error"/>
  </state>
  <state id="check">
    <transition cond="DN=='7102'" target="routing"/>
    <transition target="error"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Fetch Data in JSON Format

As the following sample demonstrates, the fetched resource can return data in JSON format. The data can then be parsed by an SCXML session. Note that in this example the result of the parsing is stored in the Target global variable.

Fetched file content: File http://localhost:9090/strategies/01_BASIC/_aux/GetDate.jsp

```
<%="{\"year\":2008, \"month\":\"December\", \"date\":25}"%>
```

Root strategy

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:session="www.genesyslab.com/modules/session"
        initial="initial">
  <script>var date = ""</script>
  <state id="initial">
    <transition event="interaction.added" target="fetch"/>
  </state>
  <state id="fetch">
    <onentry>
      <script>var URI="http://localhost:9090/strategies/01_BASIC/_aux/GetDate.jsp";</script>
      <session:fetch srcexpr="URI" type="'application/json'"/>
    </onentry>
    <transition event="session.fetch.done" target="check">
      <script>date = JSON.parse(_event.data.content);</script>
    </transition>
    <transition event="error.session.fetch" target="error"/>
  </state>
  <state id="check" >
    <transition cond="date.month=='December' && date.date==25" target="routing"/>
    <transition target="error"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>
    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>
  <final id="exit"/>
  <final id="error"/>
```

```
</scxml>
```

Fetch Data With Parameters

The following sample uses the TargetType parameter when fetching data, providing additional information that is used on the Web Server side to calculate the target.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:session="www.genesyslab.com/modules/session"
        initial="initial">
  <script>var DN = "";</script>

  <state id="initial">
    <transition event="interaction.added" target="fetch"/>
  </state>
  <state id="fetch">
    <onentry>
      <script>var URI="http://localhost:9090/strategies/01_BASIC/_aux/
GetDNWithParam.jsp";</script>
      <session:fetch srcexpr="URI">
        <param name="DN" expr="'7102'"/>
      </session:fetch>
    </onentry>
    <transition event="session.fetch.done" target="check">
      <script>DN = _event.data.content;</script>
    </transition>
    <transition event="error.session.fetch" target="error"/>
  </state>
  <state id="check" >
    <transition cond="DN=='7102'" target="routing"/>
    <transition target="error"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Fetch Data Using POST Method

The SCXML session can use the POST HTTP request to fetch data from a Web Server, as opposed to the GET request that is used by default. The following sample shows how this can be achieved.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:session="www.genesyslab.com/modules/session"
        initial="initial">
  <script>var DN = "";</script>
  <state id="initial">
    <transition event="interaction.added" target="fetch"/>
  </state>
  <state id="fetch">
    <onentry>
      <script>var URI="http://localhost:9090/strategies/01_BASIC/_aux/
GetDNOnPOST.jsp";</script>
      <session:fetch srcexpr="URI" method="'post'"/>
    </onentry>
    <transition event="session.fetch.done" target="check">
      <script>DN = _event.data.content;</script>
    </transition>
    <transition event="error.session.fetch" target="error"/>
  </state>
  <state id="check" >
    <transition cond="DN=='7102'" target="routing"/>
    <transition target="error"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Handle Fetch Failure

The following sample demonstrates how to route to the default DN when the fetch request fails.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:session="www.genesyslab.com/modules/session"
        initial="initial">
  <state id="initial">
    <onentry>
      <session:fetch srcexpr="'http://localhost:9090/strategies/DN2.jsp'"/>
    </onentry>
    <transition event="session.fetch.done" target="error"/>
    <transition event="error.session.fetch" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Invoke Session

An SCXML strategy can invoke another SCXML strategy, as the following sample demonstrates.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="invocation"/>
  </state>
  <script>
    var URI = "http://localhost:9090/strategies/01_BASIC/_aux/RouteToDN.xml";
  </script>
  <state id="invocation">
    <invoke src="URI" type="scxml">
      <param name="ixnid" expr="_genesys.ixn.firstixnid"/>
    </invoke>
    <transition event="done.invoke.invocation.*" target="exit">
      <log expr="'DONE'"/>
    </transition>
    <transition event="error.invoke.invocation.*" target="error"/>
  </state>

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

The invoked RouteToDN.xml strategy is similar to the one presented here:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <datamodel>
    <data ID="ixnid"/>
  </datamodel>
  <state id="initial">
    <transition target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit interactionid="_data.ixnid" priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetobjectselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>
  <final id="exit"/>
  <final id="error"/>
</scxml>
```

Invoke Session With Parameters

An SCXML session can use parameters to pass additional information to the SCXML session being invoked, as shown in the following example.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="invocation"/>
  </state>
  <state id="invocation">
    <invoke src="'http://localhost:9090/strategies/01_BASIC/_aux/RouteToDNWithParam.xml'"
type="scxml">
      <param name="DN" expr="'7102'"/>
      <param name="ixnid" expr="_genesys.ixn.firstixnid"/>
    </invoke>
    <transition event="done.invoke.invocation.*" target="exit">
      <log expr="'DONE'"/>
    </transition>
    <transition event="error.invoke.invocation.*" target="error"/>
  </state>

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

RouteToDNWithParam.xml makes use of the parameter in the following way:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <datamodel>
    <data id="DN"/>
    <data ID="ixnid"/>
  </datamodel>

  <state id="initial">
    <transition target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit interactionid="_data.ixnid" priority="5" timeout="20">
        <queue:targets>
          <queue:target type="dn" name="_data.DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Receive Event From Invoked Session

The invoked SCXML session can communicate with the invoking session by sending an event back to the parent session and passing additional information in the event.

The following sample stores the information passed from the invoked session using an empty finalize element.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="invocation"/>
  </state>
  <datamodel>
    <data ID="DN" expr="''" />
  </datamodel>
  <state id="invocation">
    <invoke src="'http://localhost:9090/strategies/01_BASIC/_aux/ReturnDN.xml'" type="scxml">
      <finalize/>
    </invoke>
    <transition event="done.invoke.invocation.*" target="routing"/>
    <transition event="error.invoke.invocation.*" target="error"/>
  </state>
  <state id="routing">
    <onentry>
       <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="_data.DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Here is a sample for ReturnDN.xml, which is invoked above:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition target="event"/>
  </state>
  <datamodel>
    <data ID="DN" expr="'7102'"/>
  </datamodel>
  <state id="event">
    <onentry>
      <send event="'DN'" target="'_parent'" type="'scxml'" >
         <param name="DN" expr="_data.DN" />
      </send>
    </onentry>

    <transition target="exit"/>
  </state>

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

Receive Event From Invoked Session and Extract Event Data

The invoking session can extract data from the event sent by the invoked session using the <finalize> clause, as shown in the following sample.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <datamodel>
    <data id="DN"/>
  </datamodel>
  <state id="initial">
    <transition event="interaction.added" target="invocation"/>
  </state>
  <state id="invocation">
    <invoke
    src="'http://localhost:9090/strategies/01_BASIC/_aux/ReturnDN.xml'" type="scxml">
      <finalize>
        <script>if ( _event.name == 'DN' ) _data.DN=_event.data.DN; </script>
      </finalize>
    </invoke>
    <transition event="done.invoke.invocation.*" target="routing"/>
    <transition event="error.invoke.invocation.*" target="error"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="_data.DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Cancel Invoked Session

The invoked SCXML session is cancelled if the invoking session exits the state where the invocation is made without waiting for an invocation-completion event, as shown in the sample below.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="invocation"/>
  </state>
  <state id="invocation">
    <invoke
    src="'http://localhost:9090/strategies/01_BASIC/_aux/CheckInvokeCancel.xml'"
    type="scxml"/>
    <transition event="invoked" target="routing"/>
    <transition event="error.invoke.invocation.*" target="error"/>
  </state>

  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Handle Invoke Failure

The following sample demonstrates how to route to the default DN when the invoke request fails.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="invocation"/>
  </state>
  <state id="invocation">
    <invoke
    src="'http://localhost:9090/strategies/01_BASIC/_aux/ReturnDN.xml'"
    type="scxml">
      <finalize/>
    </invoke>
    <transition event="done.invoke.invocation.*" target="routing"/>
    <transition event="error.invoke.invocation.*" target="default"/>
  </state>
  <state id="routing">
    <onentry>
       <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="_data.DN"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

  <state id="default">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'1100'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="default'"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>
  <final id="exit"/>
  <final id="error"/>
</scxml>
```

Access Interaction Properties

The following SCXML strategy shows how to access the properties of an interaction object.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:ixn="http://www.genesyslab.com/modules/interaction"
        initial="initial">
  <script>
        var ixnid;
  </script>

  <state id="initial">
        <transition event="interaction.added" target="get_ixn_properties">
                 <script>
                        ixnid = _event.data.interactionid;
                </script>
        </transition>
  </state>
  <state id="get_ixn_properties">
        <onentry>
                <script>
                        var g_uid = _genesys.ixn.interactions[ixnid].g_uid;
                        var category = _genesys.ixn.interactions[ixnid].category;
                        var tenantid = _genesys.ixn.interactions[ixnid].tenantid;
                        var parentid = _genesys.ixn.interactions[ixnid].parentid;
                        var contactedaddr = _genesys.ixn.interactions[ixnid].contactedaddr;
                        var parties = _genesys.ixn.interactions[ixnid].parties;
                        var udata = _genesys.ixn.interactions[ixnid].udata;
                        var voice = _genesys.ixn.interactions[ixnid].voice;
                        var xdata = _genesys.ixn.interactions[ixnid].xdata;
                        var location = _genesys.ixn.interactions[ixnid].location;
                </script>
        </onentry>

        <transition cond="category=='voice'" target="exit">
                <log expr="'g_uid: ' + g_uid" />
                <log expr="'category: ' + category" />
                <log expr="'tenantid: ' + tenantid" />
                <log expr="'parentid: ' + parentid" />
                <log expr="'contactedaddr: ' + contactedaddr" />
                <log expr="'parties: ' + uneval(parties)" />
                <log expr="'udata: ' + uneval(udata)" />
                <log expr="'voice: ' + uneval(voice)" />
                <log expr="'xdata: ' + uneval(xdata)" />
                <log expr="'location: ' + uneval(location)" />
        </transition>

        <transition target="error" />
  </state>

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

Set User Data

The following SCXML strategy shows how to set and update user data for an interaction.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:ixn="http://www.genesyslab.com/modules/interaction"
        initial="initial">

  <state id="initial">
        <transition event="interaction.added" target="setudata" >
                <script>
                        _data.ixnid = _event.data.interactionid;
                </script>
        </transition>
  </state>
  <state id="setudata">
        <onentry>
                <script>
                        var data = { details : { name : "Smith, John", age : 45 } };
                        _genesys.ixn.setuData( data );
                        _genesys.ixn.setuData( { category : 1 } );
                </script>
        </onentry>
        <transition target="update"/>
  </state>
  <state id="update">
        <onentry>
                <script>
                        _genesys.ixn.setuData( { category : 2 }, _data.ixnid );
                </script>
        </onentry>
        <transition target="check"/>
  </state>

  <state id="check">
        <transition event="interaction.udata.changed"
                cond="_genesys.ixn.interactions[_data.ixnid].udata.category==2 &&
                _genesys.ixn.interactions[_data.ixnid].udata.details.name=='Smith, John' &&
                _genesys.ixn.interactions[_data.ixnid].udata.details.age==45"
                target="exit" />
        <transition target="error" />
  </state>
  <final id="exit"/>
  <final id="error"/>
</scxml>
```

Delete User Data

The following SCXML strategy shows how to delete a property from the user data of an interaction.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:ixn="http://www.genesyslab.com/modules/interaction"
        initial="initial">
  <state id="initial">
        <transition event="interaction.added" target="setudata">
                <script>
                        _data.ixnid = _event.data.interactionid;
                </script>
        </transition>
  </state>
  <state id="setudata">
        <onentry>
                <script>
                        var data = { details : { name : "Smith, John", age : 45 } };
                        _genesys.ixn.setuData( data );
                </script>
        </onentry>
        <transition event="interaction.udata.changed"
                cond="_genesys.ixn.interactions[_data.ixnid].udata.details.name=='Smith,
John' &&
                _genesys.ixn.interactions[_data.ixnid].udata.details.age==45"
                target="delete"/>
  </state>
  <state id="delete">
        <onentry>
                <script>
                        _genesys.ixn.deleteuData( { details:{ age:0 } }, _data.ixnid );
                </script>
        </onentry>
        <transition event="interaction.udata.changed"
                cond="_genesys.ixn.interactions[_data.ixnid].udata.details.name=='Smith,
John' &&
                _genesys.ixn.interactions[_data.ixnid].udata.details.age==undefined"
                target="exit"/>
        <transition target="error"/>
  </state>
  <final id="exit"/>
  <final id="error"/>
</scxml>
```

Delete All User Data

The following SCXML strategy shows how to delete all user data from an interaction.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:ixn="http://www.genesyslab.com/modules/interaction"
        initial="initial">
  <state id="initial">
        <transition event="interaction.added" target="setudata">
                <script>
                        _data.ixnid = _event.data.interactionid;
                </script>
        </transition>
  </state>
  <state id="setudata">
        <onentry>
                <script>
                        var data = { details : { name : "Smith, John", age : 45 } };
                        _genesys.ixn.setuData( data, _data.ixnid );
                </script>
        </onentry>
        <transition event="interaction.udata.changed"
                cond="_genesys.ixn.interactions[_data.ixnid].udata.details.name=='Smith,
John' &&
                _genesys.ixn.interactions[_data.ixnid].udata.details.age==45"
                target="delete_all_udata"/>
  </state>

  <state id="delete_all_udata">
        <onentry>
                <script>
                        _genesys.ixn.deleteuData( "$ALL", _data.ixnid );
                </script>
        </onentry>
        <transition event="interaction.udata.changed"
                cond="_genesys.ixn.interactions[_data.ixnid].udata.ORSession==undefined &&
                _genesys.ixn.interactions[_data.ixnid].udata.ORDbid==undefined &&
                _genesys.ixn.interactions[_data.ixnid].udata.ORUrl==undefined &&
                _genesys.ixn.interactions[_data.ixnid].udata.details==undefined"
                target="exit"/>
        <transition target="error"/>
  </state>

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

Access Voice Interaction Properties

The following SCXML strategy shows how to access the properties of a voice interaction object.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:ixn="http://www.genesyslab.com/modules/interaction"
        initial="initial">
  <script>
        var ixnid;
  </script>

  <state id="initial">
        <transition event="interaction.added" target="get_ixn_voice_properties">
                 <script>
                        ixnid = _event.data.interactionid;
                </script>
        </transition>
  </state>
  <state id="get_ixn_voice_properties">
        <onentry>
                <script>
                        var type = _genesys.ixn.interactions[ixnid].voice.type;
                        var media = _genesys.ixn.interactions[ixnid].voice.media;
                        var ani = _genesys.ixn.interactions[ixnid].voice.ani;
                        var dnis = _genesys.ixn.interactions[ixnid].voice.dnis;
                        var acdq = _genesys.ixn.interactions[ixnid].voice.acdq;
                        var callid = _genesys.ixn.interactions[ixnid].voice.callid;
                        var connid = _genesys.ixn.interactions[ixnid].voice.connid;
                </script>
        </onentry>

        <transition cond="type=='internal' && media=='TMediaVoice'" target="exit">
                <log expr="'type: ' + type" />
                <log expr="'media: ' + media" />
                <log expr="'ani: ' + ani" />
                <log expr="'dnis: ' + dnis" />
                <log expr="'acdq: ' + acdq" />
                <log expr="'callid: ' + callid" />
                <log expr="'connid: ' + connid" />
        </transition>

        <transition target="error" />
  </state>

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

Create Call

The following SCXML strategy shows how to create a voice call with extension hints.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:ixn="http://www.genesyslab.com/modules/interaction"
        initial="initial">
  <datamodel>
    <data ID="reqid"/>
    <data ID="ixnid"/>
    <data ID="time_delay" expr="'2s'" />
  </datamodel>
  <state id="initial">
    <transition event="interaction.added" target="calling"/>
  </state>
  <state id="calling">
    <onentry>
      <log expr="'Calling ...'" />
      <ixn:createcall requestid="_data.reqid" type="regular" from="'701'" to="'702'"
udata="({key_1:10*10, key_2:20*20, key_3:30*30})" hints="({extensions:{key_1:100*100,
key_2:200*200, key_3:300*300}})" />
    </onentry>
    <transition event="error.voice.createcall" cond="_event.data.requestid == _data.reqid"
target="error">
      <log expr="'Got createcall error:'" />
      <log expr="uneval( _event )" />
    </transition>
    <transition event="voice.createcall.done" cond="_event.data.requestid == _data.reqid"
target="delay">
      <log expr="'Got createcall confirmation:'"/>
      <log expr="'event name = ' + _event.name" />
      <script>
        _data.ixnid = _event.data.interactionid;
      </script>
    </transition>
  </state>
  <state id="delay">
    <onentry>
      <log expr="'======== Inside Delay ========'"/>
      <send event="'SynchroEvent'" delay="_data.time_delay"/>
    </onentry>
    <transition event="SynchroEvent" target="acceptcall" />
  </state>
  <state id="acceptcall">
    <onentry>
      <ixn:accept requestid="_data.reqid" interactionid="_data.ixnid" resource="'702'" />
    </onentry>
    <transition event="error.interaction.accept" target="error" />
    <transition event="interaction.accept.done" cond="_event.data.requestid == _data.reqid"
target="exit" />
  </state>
  <final id="exit"/>
  <final id="error"/>
</scxml>
```

Collect Digits

The following SCXML strategy collects a set of digits from a caller. The digits collected is returned in the `_event.data.digits` object and is stored in the user data.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
        <transition event="interaction.added" target="dialog">
                <script>
                        _data.ixnid = _event.data.interactionid;
                </script>
        </transition>
  </state>
  <state id="dialog">
        <onentry>
                <dialog:collect>
                        <dialog:input max_digits="6" abort_digits="'1'" term_digits="'9'"
total_timeout="20" ignore_digits="'hj'"
                        backspace_digits = "'kl'" reset_digits="'123'" clear="false"
start_timeout="10" digit_timeout="5"/>
                </dialog:collect>
        </onentry>
        <transition event="dialog.collect.done" target="exit">
                <script>
                        _genesys.ixn.setuData({ "CED" : _event.data.digits }, _data.ixnid );
                </script>
        </transition>
        <transition event="error.dialog.collect" target="error"/>
  </state>

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

Play Announcement With One Prompt

The following SCXML strategy plays an announcement to a caller.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
        <transition event="interaction.added" target="dialog"/>
  </state>
  <state id="dialog">
        <onentry>
                <dialog:play language="'English(US)'">
                        <dialog:prompts type="ann">
                                <dialog:prompt interrupt="true" intid="1"/>
                        </dialog:prompts>
                </dialog:play>
        </onentry>
        <transition event="dialog.play.done" target="routing"/>
        <transition event="error.dialog.play" target="error"/>
  </state>

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

Play Announcement With Two Prompts

The following SCXML strategy plays an announcement with two prompts.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
        <state id="initial">
                <transition event="interaction.added" target="dialog" />
        </state>
        <state id="dialog">
                <onentry>
                        <dialog:play language="'English(US)'">
                                <dialog:prompts type="ann">
                                        <dialog:prompt interrupt="true" intid="1" />
                                        <dialog:prompt interrupt="true" intid="2" />
                                </dialog:prompts>
                        </dialog:play>
                </onentry>
                <transition event="dialog.play.done" target="exit" />
                <transition event="error.dialog.play" target="error" />
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Play Announcement and Collect Digits

The following SCXML strategy plays an announcement to, and collects digits from, a caller.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
        <state id="initial">
                <transition event="interaction.added" target="dialog" />
        </state>
        <state id="dialog">
                <onentry>
                        <dialog:playandcollect>
                                <dialog:prompts type="ann">
                                        <dialog:prompt interrupt="true" intid="1111" />
                                        <dialog:prompt interrupt="true" number="'2222'" />
                                </dialog:prompts>
                                <dialog:input max_digits="6" abort_digits="'1'"
                                        term_digits="'9'" total_timeout="30" start_timeout="5"
                                        digit_timeout="5" />
                        </dialog:playandcollect>
                </onentry>
                <transition event="dialog.playandcollect.done"
cond="_event.data.digits=='422678'"
                        target="exit" />
                <transition event="error.dialog.playandcollect" target="error" />
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Play Sound

The following SCXML strategy plays a voice-related sound to a caller.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue" xmlns:dialog="www.genesyslab.com/
modules/dialog"
        initial="initial">
        <state id="initial">
                <transition event="interaction.added" target="dialog" />
        </state>
        <state id="dialog">
                <onentry>
                        <dialog:playsound type="'busy'" duration="10" />
                </onentry>
                <transition event="dialog.playsound.done" target="exit" />
                <transition event="error.dialog.playsound" target="error" />
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Record User Announcement

The following SCXML strategy creates and records an announcement from a caller.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
        <state id="initial">
                <transition event="interaction.added" target="dialog" />
        </state>
        <state id="dialog">
                <onentry>
                        <dialog:createann userid="'12334567'" abort_digits="'1'"
                                term_digits="'9'" total_timeout="20" reset_digits="'123'"
                                start_timeout="456">
                                <dialog:prompts type="ann">
                                        <dialog:prompt interrupt="true" intid="1111" />
                                </dialog:prompts>
                        </dialog:createann>
                </onentry>
                <transition event="dialog.createann.done" target="exit" />
                <transition event="error.dialog.createann" target="error" />
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Delete User Announcement

The following sample demonstrates how to delete an announcement from a strategy.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
        <state id="initial">
                <transition event="interaction.added" target="dialog" />
        </state>
        <state id="dialog">
                <onentry>
                        <dialog:deleteann userid="'12334567'" annid="464" />
                </onentry>
                <transition event="dialog.deleteann.done" target="exit" />
                <transition event="error.dialog.deleteann" target="error" />
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Play Application

The following SCXML strategy requests that a specific dialog be started on a specified interaction by a specified resource.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
        <state id="initial">
                <transition event="interaction.added" target="dialog" />
        </state>
        <state id="dialog">
                <onentry>
                        <dialog:start type="'applid'" application="464646464" />
                </onentry>
                <transition event="dialog.start.done" target="exit" />
                <transition event="error.dialog.start" target="error" />
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Cancel Call

The following sample demonstrates how to cancel a currently running dialog application.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
        <state id="initial">
                <transition event="interaction.added" target="dialog" />
        </state>
        <state id="dialog">
                <onentry>
                        <dialog:stop compatible="true" />
                </onentry>
                <transition event="dialog.stop.done" target="exit" />
                <transition event="error.dialog.stop" target="error" />
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Start on Remote Resource

The following SCXML strategy requests that a specific dialog be started on a specified interaction by a new remote resource.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
        <state id="initial">
                <transition event="interaction.added" target="dialog" />
        </state>
        <state id="dialog">
                <onentry>
                        <dialog:remote destination="'123456'" default="'2334'" />
                </onentry>
                <transition event="dialog.remote.done" target="exit" />
                <transition event="error.dialog.remote" target="error" />
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Run Series of Treatments

The following SCXML strategy performs a series of treatments in the order specified.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
        <state id="initial">
                <transition event="interaction.added" target="routing" />
        </state>
        <state id="routing">
                <onentry>
                        <queue:submit priority="5" timeout="20">
                                <queue:targets>
                                        <queue:target type="agent" name="'az'" />
                                </queue:targets>
                                <dialog:runtreatments>
                                        <dialog:play language="'English_US'">
                                                <dialog:prompts type="ann">
                                                        <dialog:prompt interrupt="true"
text="'You reached Genesys'" />
                                                </dialog:prompts>
                                        </dialog:play>
                                </dialog:runtreatments>
                        </queue:submit>
                </onentry>
                <transition event="queue.submit.done" target="exit">
                        <log expr="'DONE'" />
                        <log expr="_genesys.ixn.interactions[0].voice.ani" />
                        <log expr="'DONE'" />
                        <log expr="_event.data.targetselected" />
                </transition>
                <transition event="error.queue.submit" target="error" />
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Get Statistic Value

The following example requests a statistic in the script block and then uses a delay to determine when this value will become available to the SCXML session.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:statistic="www.genesyslab.com/modules/statistic"
        initial="initial">
  <state id="initial">
        <transition event="interaction.added" target="subscribe"/>
  </state>
  <state id="subscribe">
        <onentry>
                <statistic:subscribe object="'SipGr_2@.GA'"
statistic="'StatAgentsAvailable'"/>
        </onentry>

        <transition event="statistic.subscribe.done" target="delay"/>
        <transition event="error.statistic.subscribe" target="error"/>
  </state>
  <state id="delay">
        <onentry>
                <send event="'delay'" target="_sessionid" targettype="'scxml'" delay="'2s'"/>
        </onentry>
        <transition event="delay" target="check"/>
  </state>
  <state id="check">
        <transition cond="_genesys.statistic.sData('SipGr_2@.GA', 'StatAgentsAvailable')==1"
target="exit"/>
        <transition cond="_genesys.statistic.sData('SipGr_2@.GA', 'StatAgentsAvailable')!=1"
target="error"/>
  </state>

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

Get Average Statistic Value

The following example requests the average statistic value for a set of targets in the script block and then uses a delay to indicate when this value will become available to the SCXML session.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:statistic="www.genesyslab.com/modules/statistic"
        initial="initial">
  <state id="initial">
        <transition event="interaction.added" target="subscribe"/>
  </state>
  <state id="subscribe">
        <onentry>
                <statistic:subscribe object="'702_sip@.A'" statistic="'StatAgentsAvailable'"/>
                <statistic:subscribe object="'SipGr_2@.GA'"
statistic="'StatAgentsAvailable'"/>
        </onentry>

        <transition event="statistic.subscribe.done" target="delay"/>
        <transition event="error.statistic.subscribe" target="error"/>
  </state>
  <state id="delay">
        <onentry>
                <send event="'delay'" target="_sessionid" targettype="'scxml'" delay="'2s'"/>
        </onentry>
        <transition event="delay" target="check"/>
  </state>
  <state id="check">
        <onentry>
                <log expr="_genesys.statistic.sData('702_sip@.A', 'StatAgentsAvailable')"/>
                <log expr="_genesys.statistic.sData('SipGr_2@.GA', 'StatAgentsAvailable')"/>
        </onentry>
        <transition cond="_genesys.statistic.getAvgData('702_sip@.A,SipGr_2@.GA',
'StatAgentsAvailable')==1" target="exit"/>
        <transition cond="_genesys.statistic.getAvgData('702_sip@.A,SipGr_2@.GA',
'StatAgentsAvailable')!=1" target="error"/>
  </state>

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

Get Minimum Statistic Value

The following example requests the minimum statistic value for a set of targets in the script block and then uses a delay to dictate when this value will become available to the SCXML session.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:statistic="www.genesyslab.com/modules/statistic"
        initial="initial">
  <state id="initial">
        <transition event="interaction.added" target="subscribe"/>
  </state>
  <state id="subscribe">
        <onentry>
                <statistic:subscribe object="'SipGr_2@.GA'"
statistic="'StatAgentsAvailable'"/>
                <statistic:subscribe object="'SipGr_3@.GA'"
statistic="'StatAgentsAvailable'"/>
        </onentry>

        <transition event="statistic.subscribe.done" target="delay"/>
        <transition event="error.statistic.subscribe" target="error"/>
  </state>
  <state id="delay">
        <onentry>
                <send event="'delay'" target="_sessionid" targettype="'scxml'" delay="'2s'"/>
        </onentry>
        <transition event="delay" target="check"/>
  </state>
  <state id="check">
        <transition cond="_genesys.statistic.getMinData('SipGr_2@.GA,SipGr_3@.GA',
'StatAgentsAvailable')==0" target="exit"/>
        <transition cond="_genesys.statistic.getMinData('SipGr_2@.GA,SipGr_3@.GA',
'StatAgentsAvailable')!=0" target="error"/>
  </state>

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

Get Maximum Statistic Value

The following example requests the maximum statistic value for a set of targets in the script block and then uses a delay to indicate when value will become available to the SCXML session.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:statistic="www.genesyslab.com/modules/statistic"
        initial="initial">
  <state id="initial">
        <transition event="interaction.added" target="subscribe"/>
  </state>
  <state id="subscribe">
        <onentry>
                <statistic:subscribe object="'SipGr_2@.GA'"
statistic="'StatAgentsAvailable'"/>
                <statistic:subscribe object="'SipGr_3@.GA'"
statistic="'StatAgentsAvailable'"/>
        </onentry>

        <transition event="statistic.subscribe.done" target="delay"/>
        <transition event="error.statistic.subscribe" target="error"/>
  </state>
  <state id="delay">
        <onentry>
                <send event="'delay'" target="_sessionid" targettype="'scxml'" delay="'2s'"/>
        </onentry>
        <transition event="delay" target="check"/>
  </state>
  <state id="check">
        <transition cond="_genesys.statistic.getMinData('SipGr_2@.GA,SipGr_3@.GA',
'StatAgentsAvailable')==0" target="exit"/>
        <transition cond="_genesys.statistic.getMinData('SipGr_2@.GA,SipGr_3@.GA',
'StatAgentsAvailable')!=0" target="error"/>
  </state>

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

Get Configuration Option Name

The following SCXML strategy uses the getConfigOption function from the session module.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="check"/>
  </state>
  <state id="check">
    <transition
        cond="_genesys.session.getConfigOption(_genesys.ixn.interactions[0].g_uid,
        'test',
        _genesys.session.lookupseq.StartFromCDN)=='12345'"
        target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Check If Special Day

The following SCXML strategy uses the isSpecialDay function from the session module.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="check"/>
  </state>
  <state id="check">
    <transition cond="_genesys.session.isSpecialDay('Call Center Open',
    'Any Day', '', false)=='true'" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Get List Item Value

The following SCXML strategy uses the getListItemValue function from the session module.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="check"/>
  </state>
  <state id="check">
    <transition cond="_genesys.session.getListItemValue('Call In Numbers',
    'MasterCard', 'number')=='18002343434'" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Lookup Value

The following SCXML strategy uses the listLookupValue function from the session module.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="check"/>
  </state>
  <state id="check">
    <transition cond="_genesys.session.listLookupValue('Call In Numbers',
    'Discover')=='true'" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Get Time in Time Zone

The following SCXML strategy uses the timeInZone function from the session module.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="check"/>
  </state>
  <state id="check">
    <transition cond="_genesys.session.timeInZone('PST')" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.session.timeInZone('PST')"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Get Date in Time Zone

The following SCXML strategy uses the dateInZone function from the session module.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="check"/>
  </state>
  <state id="check">
    <transition cond="_genesys.session.dateInZone('PST')" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="Date.parse(_genesys.session.dateInZone('PST'))"/>
      <log expr="_genesys.session.dateInZone('PST')"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Get Day in Time Zone

The following SCXML strategy uses the dayInZone function from the session module.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        initial="initial">
  <state id="initial">
    <transition event="interaction.added" target="check"/>
  </state>
  <state id="check">
    <transition cond="_genesys.session.dayInZone('PST')" target="routing"/>
  </state>
  <state id="routing">
    <onentry>
      <queue:submit priority="5" timeout="20">
        <queue:targets type="dn">
          <queue:target name="'7102'"/>
        </queue:targets>
      </queue:submit>
    </onentry>

    <transition event="queue.submit.done" target="exit">
      <log expr="'DONE'"/>
      <log expr="_genesys.ixn.interactions[0].voice.ani"/>
      <log expr="'DONE'"/>
      <log expr="_event.data.targetselected"/>
    </transition>
    <transition event="error.queue.submit" target="error"/>
  </state>

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

Work With E-Mail Or SMS

The following SCXML strategy receives an inbound e-mail or SMS from a customer. It then sends an acknowledgement is to the customer and calls submit to find an available agent. The e-mail or SMS is redirected to the agent and an e-mail response from the agent is sent to the customer. Any inbound e-mail delivery confirmation is also handled.

```
<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:ws="http://www.genesyslab.com/modules/ws"
        xmlns:dialog="http://www.genesyslab.com/modules/dialog"
        _persist="false" >
        <datamodel>
                <data ID="reqid" expr="''"/>
                <data ID="childIxn" expr="''"/>
                <data ID="routetarget" expr="''"/>
        </datamodel>
        <initial>
                <transition target="listening">
                </transition>
        </initial>
        <state id="main" initial="listening">
                <state id="listening">
                        <onentry>
                                <log expr="'Listening ...'" />
                        </onentry>
                        <!-- This script handles email and sms interactions -->
                        <transition event="interaction.present"
cond="_genesys.ixn.interactions[_event.data.interactionid].category =='msgbased'"
target="arriving_ixn">
                                <script>
                                        _data.ixnid = _event.data.interactionid;
                                </script>
                        </transition>
                </state>
                <state id="arriving_ixn">
                        <onentry>
                                      <log expr="uneval( global )" />
                                <if
cond="_genesys.ixn.interactions[_data.ixnid].msgbased.media == 'TMediaEMail'">
                                        <if
cond="_genesys.ixn.interactions[_data.ixnid].udata.FromAddress ==
'postmaster@genesyslab.com'">
                                                <!-- This looks to be a delivery
confirmation.  -->
                                                <event name="handle.email.confirmation"/>
                                        <else/>
                                                <event name="send.email.ack"/>
                                        </if>
                                <elseif
cond="_genesys.ixn.interactions[_data.ixnid].msgbased.media == 'TMediaNativeSMS'"/>
                                        <event name="send.sms.ack"/>
                                </if>
                        </onentry>
                        <transition event="handle.email.confirmation" target="exit">
                                <!-- Add logic here to handle the delivery confirmation... -->
                                <!-- Terminate the confirmation interaction -->
                                <ixn:terminate interactionid="_data.ixnid" />
                        </transition>
                        <transition event="send.email.ack">
                                <!-- Send an Email acknowledgment -->
                                <!-- Acknowledgment text is a "Standard Response" with the
specified, 16-digit name. -->
                                <ixn:createmessage requestid="_data.reqid"
type="acknowledgement"
                                        media="_genesys.ixn.mediaType.TMediaEMail"
to="'_origin'" subject="'$USESRL'"
                                        msgsrc="'gdata:config\\SR.0000Na5C1F3500JW'"
delivery="false" />
                        </transition>
                        <transition event="send.sms.ack">
                                <!-- Send an SMS acknowledgment -->
                                 <ixn:createmessage requestid="_data.reqid"
type="acknowledgement"
                                        media="_genesys.ixn.mediaType.TMediaNativeSMS"
to="'_udata\\_smsSrcNumber'" from="'8881234567'"
                                        msgsrc="'Your request has been received and is being
processed'" />
                        </transition>
                        <transition event="msgbased.createmessage.done" target="find_agent"/>
                        <transition event="error.msgbased.createmessage" target="ack_failed"/>
                </state>
                <state id="ack_failed">
                        <transition  target="find_agent">
                                <log expr="Acknowledgement request failed and should be
handled here." />
                        </transition>
                </state>
                <state id="find_agent">
                        <onentry>
                                <!-- Submit URS request to find an agent. -->
                                <queue:submit route="false">
                                        <queue:targets>
                                                <queue:target name="'5'" type="agent"
statserver="'statserver'" />
                                                <queue:target name="'6'" type="agent"
statserver="'statserver'" />
                                                <queue:target name="'7'" type="agent"
statserver="'statserver'" />
                                        </queue:targets>
                                </queue:submit>
                        </onentry>
                        <transition event="queue.submit.done" target="deliver_to_agent">
                                <log expr="uneval( _event )" />
                                <assign location="_data.routetarget"
expr="_event.data.resource" />
                        </transition>
                        <transition event="error.queue.submit" target="error">
                                <log expr="uneval( _event )" />
                        </transition>
                </state>
                <state id="deliver_to_agent">
                        <onentry>
                                <log expr="'Routing ...'" />
                                <script>
                                        var hint = new Object();
                                        hint.outqueues = new Object();
                                        hint.outqueues.Orchestration_queue = "queue for
outbound response";
                                </script>
                                <!-- For redirect, the 'to' object is gotten from data
returned from queue:submit -->
                                <ixn:redirect requestid="_data.reqid"
interactionid="_data.ixnid"
                                        from="'scxml'" to="_data.routetarget" hints="hint"/>
                        </onentry>
                        <transition event="interaction.redirect.done" target="responding">
                                <log expr="uneval( _event )" />
                        </transition>
                        <transition event="error.interaction.redirect" target="error">
                                <log expr="uneval( _event )" />
                        </transition>
                </state>
                <parallel id ="responding">
                        <onentry>
                                <log expr="'Process Agent response...'" />
                        </onentry>

                        <state id="responding_outbound">
                                <transition event="interaction.present"
cond="_genesys.ixn.interactions[_event.data.interactionid].msgbased.type == 'OutboundReply'">
                                        <!-- This present occurs if agent sends outbound
response to queue which this ORS instance manages -->
                                        <log expr="'Event Interaction Present...'" />
                                        <assign location="_data.childIxn"
expr="_event.data.interactionid" />
                                        <!-- Send agent's response to customer -->
                                        <ixn:sendmessage interactionid="_data.childIxn"
delivery="true"/>
                                   </transition>
                                <transition event="msgbased.sendmessage.done"
target="cleanup">
                                        <log expr="'Sending of outbound response
successful.'" />
                                        <ixn:terminate interactionid="_data.childIxn"
/>
                                </transition>
                                <transition event="error.msgbased.sendmessage"
target="requesterror">
                                        <log expr="'Error in sending outbound response.'" />
                                        <log expr="uneval( _event )" />
                                </transition>
                        </state>
                        <state id="responding_initial_interaction">
                                <transition event="interaction.notcontrolled">
                                        <log expr="'Initial interaction is no longer
controlled by this scxml.'" />
                                </transition>
                                <transition event="interaction.deleted">
                                        <!-- Initial interaction is going away -->
                                        <!-- Wait up to 60 seconds for any response to be
processed (in parallel state). -->
                                        <send delay="'60s'" type="'scxml'"
event="'wait_is_over'" />
                                </transition>
                                <transition event="wait_is_over" target="cleanup" />
                        </state>
                </parallel>
                <state id="cleanup">
                        <onentry>
                                <log expr="'Cleanup logic goes here...'" />
                        </onentry>
                        <transition target="exit"/>
                </state>
                <state id="requesterror">
                        <onentry>
                                <log expr="'Request error processing goes here...'" />
                        </onentry>
                        <transition target="error"/>
                </state>
        </state>
        <final id="error"/>
        <final id="exit"/>
</scxml>
```

Work With Chat

The following SCXML strategy receives an inbound chat from the customer. Submit is called to find an available agent and the chat interaction is redirected to the agent. Upon termination of the chat interaction, a transcript of the chat session is e-mailed to the customer.

```
<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:ws="http://www.genesyslab.com/modules/ws"
        xmlns:dialog="http://www.genesyslab.com/modules/dialog"
        _persist="false" >
        <datamodel>
                <data ID="reqid" expr="''"/>
                <data ID="ixnid" expr="''"/>
                <data ID="routetarget" expr="''"/>
        </datamodel>
        <initial>
                <transition target="listening">
                </transition>
        </initial>
        <state id="main" initial="listening">
                <state id="listening">
                        <onentry>
                                <log expr="'Listening ...'" />
                        </onentry>
                        <!-- This script handles chat interactions -->
                        <transition event="interaction.present"
cond="_genesys.ixn.interactions[_event.data.interactionid].category =='chat'"
target="find_agent">
                                <script>
                                        _data.ixnid = _event.data.interactionid;
                                </script>
                        </transition>
                </state>
                <state id="find_agent">
                        <onentry>
                                       <log expr="uneval( global )" />
                                <!-- Submit URS request to find an agent. -->
                                <queue:submit route="false">
                                        <queue:targets>
                                                <queue:target name="'3'" type="agent"
statserver="'statserver'" />
                                                <queue:target name="'4'" type="agent"
statserver="'statserver'" />
                                                <queue:target name="'5'" type="agent"
statserver="'statserver'" />
                                        </queue:targets>
                                </queue:submit>
                        </onentry>
                        <transition event="queue.submit.done" target="deliver_to_agent">
                                <log expr="uneval( _event )" />
                                <assign location="_data.routetarget"
expr="_event.data.resource" />
                        </transition>
                        <transition event="error.queue.submit" target="error">
                                <log expr="uneval( _event )" />
                        </transition>
                </state>
                <state id="deliver_to_agent">
                        <onentry>
                                <log expr="'Routing ...'" />
                                <script>
                                        var hint = new Object();
                                        hint.outqueues = new Object();
                                        hint.outqueues.Orchestration_queue = "queue for
outbound response";
                                </script>
                                <!-- For redirect, the 'to' object is gotten from data
returned from queue:submit -->
                                <ixn:redirect requestid="_data.reqid"
interactionid="_genesys.ixn.firstixnid"
                                        from="'scxml'" to="_data.routetarget" hints="hint"/>
                        </onentry>
                        <transition event="interaction.redirect.done" target="responding">
                                <log expr="uneval( _event )" />
                        </transition>
                        <transition event="error.interaction.redirect" target="error">
                                <log expr="uneval( _event )" />
                        </transition>
                </state>
                <state id ="responding">
                        <onentry>
                                <log expr="'Wait for interaction to be terminated and send
chat transcript...'" />
                                <script>
                                        var customer_email_address = "undefined";
                                </script>
                                <if cond="typeof
_genesys.ixn.interactions[_data.ixnid].udata.EmailAddress != 'undefined'">
                                        <assign location="customer_email_address"
expr="_genesys.ixn.interactions[_data.ixnid].udata.EmailAddress" />
                                <else/>
                                        <!-- No email address provided so continue without
sending transcript. -->
                                        <event name="no.email.address"/>
                                </if>
                        </onentry>

                        <transition event="interaction.notcontrolled">
                                <log expr="'Initial interaction is no longer controlled by
this scxml.'" />
                        </transition>
                        <transition event="interaction.deleted">
                                <!-- Initial interaction has been terminated -->
                                <if cond="customer_email_address != 'undefined'">
                                        <!-- Email the chat transcript.  Message text is a
standard response. -->
                                        <ixn:createmessage  type="outbound_new"
chattranscript="true"
                                                subject="'Chat transcript'"
msgsrc="'gdata:config\\SR.0000Na5C1F3500FY'"
                                                relatedixnid="_data.ixnid"
to="customer_email_address" />
                                </if>
                        </transition>
                        <transition event="msgbased.createmessage.done" target="cleanup"/>
                        <transition event="no.email.address" target="cleanup"/>
                        <transition event="error.msgbased.createmessage"
target="requesterror">
                                <log expr="uneval( _event )" />
                        </transition>
                </state>
                <state id="cleanup">
                        <onentry>
                                <log expr="'Cleanup logic goes here...'" />
                        </onentry>
                        <transition target="exit"/>
                </state>
                <state id="requesterror">
                        <onentry>
                                <log expr="'Request error processing goes here...'" />
                        </onentry>
                        <transition target="error"/>
                </state>
        </state>
        <final id="error"/>
        <final id="exit"/>
</scxml>
```

Orchestration Server Sample Templates

Using The Queue Module

- Expand Targets

Using The Interaction Interface

- Change the Ownership of an Interaction
- Detach an Interaction
- Detect Consult Call
- Detect User Data Changes

Using Multiple Interfaces

Expand Target List

The following SCXML strategy tries to route the call to various objects, starting with a specific agent. If this is not successful within 10 seconds, it will expand the target list to route the call to an agent group. Again, if this is not successful within 10 seconds, it will route the call to a place. And finally, if it is not successful within 10 seconds, it will expand the target list to a place group.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        initial="initial">
        <datamodel>
                <data id="reqid" expr="''" />
                <data id="ixnid" expr="''" />
        </datamodel>
        <state id="initial">
                <transition event="interaction.added" target="routing">
                        <script>
                                _data.ixnid = _event.data.interactionid;
                        </script>
                </transition>
        </state>
        <state id="routing" initial="route_to_agent">
                <state id="route_to_agent">
                        <onentry>
                                <log expr="'Queue Submit to Agent'" />
                                <queue:submit requestid="_data.reqid"
interactionid="_data.ixnid"
                                 priority="5" timeout="10" clearontimeout="false">
                                        <queue:targets>
                                                <queue:target type="agent" name="'701_sip'" />
                                        </queue:targets>
                                </queue:submit>
                                <send event="'to_gr1'" delay="'5s'" />
                        </onentry>
                        <transition event="error.queue.submit" target="route_to_agent_group"
/>
                </state>
                <state id="route_to_agent_group">
                        <onentry>
                                <log expr="'Queue Submit to Agent Group'" />
                                <queue:submit requestid="_data.reqid"
interactionid="_data.ixnid"
                                 priority="5" timeout="10" clearontimeout="false">
                                        <queue:targets>
                                                <queue:target type="agentgroup"
name="'SipGr_1'" />
                                        </queue:targets>
                                </queue:submit>
                        </onentry>
                        <transition event="error.queue.submit" target="route_to_place" />
                </state>
                <state id="route_to_place">
                        <onentry>
                                <log expr="'Queue Submit to Place'" />
                                <queue:submit requestid="_data.reqid"
interactionid="_data.ixnid"
                                 priority="5" timeout="10" clearontimeout="false">
                                        <queue:targets>
```

```
                                                <queue:target type="place" name="'701'" />
                                        </queue:targets>
                                </queue:submit>
                        </onentry>
                        <transition event="error.queue.submit" target="route_to_place_group"
/>
                </state>
                <state id="route_to_place_group">
                        <onentry>
                                <log expr="'Queue Submit to Place Group'" />
                                <queue:submit requestid="_data.reqid"
interactionid="_data.ixnid"
                                 priority="5" timeout="10" clearontimeout="false">
                                        <queue:targets>
                                                <queue:target type="placegroup"
name="'SIP_PlGr2'" />
                                        </queue:targets>
                                </queue:submit>
                        </onentry>
                        <transition event="error.queue.submit" target="error">
                                <log expr="'ERROR'" />
                        </transition>
                </state>
                <transition event="queue.submit.done" target="exit">
                        <log expr="'Queue Submit DONE'" />
                        <log expr="_event.data.targetselected" />
                </transition>
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Change the Ownership of an Interaction

The following example illustrates how to detach an interaction (ex. a voice call) from the parent session, and attach it to a child session that was started using `<session:start>`. The child session operates completely independently of the parent session, meaning that the termination of the parent session has no effect on the child session, and vice versa. Also, session content is not shared between the sessions.

Parent strategy

The following SCXML application detaches an interaction, then starts a new child session. The session terminates when we get a confirmation that the child session has been started.

```
<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:session="http://www.genesyslab.com/modules/session"
        initial='initial'>
        <datamodel>
                <data id="reqid" expr="''" />
                <data id="ixnid" expr="''" />
                <data id="newsessionid" expr="''" />
        </datamodel>
        <state id="initial">
                <transition event="interaction.added" target="detachcall">
                        <script>
                                _data.ixnid = _event.data.interactionid;
                        </script>
                </transition>
        </state>
        <state id="detachcall">
                <onentry>
                        <log expr="'Detach call from current session ...'" />
                        <ixn:detach requestid="_data.reqid" interactionid="_data.ixnid" />
                </onentry>
                <transition event="error.interaction.detach" target="error">
                        <log expr="'Got detach error:'" />
                        <log expr="uneval( _event )" />
                </transition>
                <transition event="interaction.detach.done"
                 cond="_event.data.requestid == _data.reqid &&
_event.data.interactionid==_data.ixnid"
                 target="create_new_session">
                        <log expr="'Got detach confirmation'" />
                        <log expr="uneval( _event )" />
                </transition>
        </state>
        <state id="create_new_session">
                <onentry>
                        <log expr="'Starting a new session that will attach the interaction
...'" />
                        <session:start src="'http://localhost:17080/ixn_attach.scxml'"
sessionid="_data.newsessionid">
                                <param name="parent_sessionid" expr="_sessionid" />
                                <param name="interactionid" expr="_data.ixnid" />
                        </session:start>
                </onentry>
                <transition event="session.start.done" target="exit">
                        <log expr="'**** Start Session Done'" />
                        <log expr="'**** Event details: ' + uneval( _event )" />
                        <log expr="'**** New Session ID: ' + _data.newsessionid" />
                </transition>
                <transition event="error.session.start" target="error">
                        <log expr="'Start Session Error'" />
                        <log expr="uneval( _event )" />
                </transition>
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Child strategy

The following SCXML application was used by the parent session to start the child session. Two parameters were passed in from the parent session, `parent_sessionid` and `interactionid` (the interactionid of the voice call that was detached from the parent session). Using the `interactionid`, the child session is able to do an `attach` and take ownership of the interaction. Then we accept the call on DN 702.

```
<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:session="http://www.genesyslab.com/modules/session"
        initial='initial'>
        <datamodel>
                <data id="reqid" expr="''" />
                <data id="ixnid" expr="''" />
                <data id="newsessionid" expr="''" />
                <data id="DN2" expr="'702'" />
                <!--    Parameters passed in from parent session
                parent_sessionid
                interactionid
                -->
        </datamodel>
        <state id="initial">
                <onentry>
                        <log expr="'New session has been started ...'" />
                        <log expr="'Datamodel = ' + uneval(_data)" />
                </onentry>
                <transition target="attachcall" />
        </state>
        <state id="attachcall">
                <onentry>
                        <log expr="'Attach call to this new session ...'" />
                        <ixn:attach requestid="_data.reqid"
interactionid="_data.interactionid" />
                </onentry>
                <transition event="error.interaction.attach" target="error">
                        <log expr="'Got attach error:'" />
                        <log expr="uneval( _event )" />
                </transition>
                <transition event="interaction.attach.done"
                 cond="_event.data.requestid == _data.reqid && _event.data.interactionid
```

```
==_data.interactionid"
                 target="acceptcall">
                        <log expr="'Got attach confirmation:'" />
                        <log expr="uneval( _event )" />
                </transition>
        </state>
        <state id="acceptcall">
                <onentry>
                        <log expr="'Answering call on dn = ' + _data.DN2" />
                        <ixn:accept requestid="_data.reqid"
interactionid="_data.interactionid"
                         resource="_data.DN2" />
                </onentry>
                <transition event="error.interaction.accept" cond="_event.data.requestid ==
_data.reqid"
                        target="error">
                        <log expr="'*** Got accept error:'" />
                        <log expr="uneval( _event )" />
                </transition>
                <transition event="interaction.accept.done"
                 cond="_event.data.requestid == _data.reqid && _event.data.interactionid ==
_data.interactionid"
                 target="exit">
                        <log expr="'*** Got accept confirmation on DN = ' + _data.DN2" />
                        <log expr="'event details = ' + uneval( _event )" />
                </transition>
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

Detach an Interaction

The following SCXML strategy detaches a call from the current session, then redirects it from DN 3001 on Switch 'tel_sw' to DN 7778 on Switch 'tel_sw_1'.

```
<!-- ****************************************** -->
<!-- * TEST OF DETACH AND SUBSEQUENT REDIRECT * -->
<!-- ****************************************** -->
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
  name="simple monitoring app"
  xmlns:queue="http://www.genesyslab.com/modules/queue"
  xmlns:ixn="http://www.genesyslab.com/modules/interaction"
  xmlns:dialog="http://www.genesyslab.com/modules/dialog" >
        <!--*******************************************************************-->
        <datamodel>
                <data ID="stateName" expr="'initial'" />
                <data ID="reqid" expr="''" />
                <data ID="ixnid" expr="''" />
                <data ID="test_result" expr="'PASSED'" />
        </datamodel>
        <!--*******************************************************************-->
        <initial>
                <transition target="waiting"/>
        </initial>
        <!--*******************************************************************-->
        <state id="main" initial="waiting">
                <!--*******************************************************************-->
                <state id="waiting">
                        <onentry>
                                <log expr="'Waiting for incoming call...'" />
                        </onentry>
                        <transition event="interaction.added" target="detachcall">
                                <script>
                                        _data.ixnid = _event.data.interactionid;
                                        var dev =
_genesys.ixn.interactions[_data.ixnid].parties[_data.ixnid + '-1'].device;
                                </script>
                                <log expr="'G_G_GOT initial interaction added: ' + uneval(
_event.data )"/>
                                <log expr="'_genesys.ixn: ' + uneval( _genesys.ixn )" />
                        </transition>
                </state>
                <!--*******************************************************************-->
                <!--  Detaching call from session with graceful termination by timeout  -->
                <!--*******************************************************************-->
                <state id="detachcall">
                        <onentry>
                                <script>
                                        var noMoreDetach = false;
                                        var detachTimeout = '3s'; /*'infinite' means endless
trying to detach*/
                                </script>
                                <if cond="detachTimeout != 'infinite'">
                                        <send event="'DetachTimeoutEvent'"
delay="detachTimeout"/>
                                </if>
                                <ixn:detach interactionid="_data.ixnid"
requestid="_data.reqid" />
                        </onentry>
                        <transition event="DetachTimeoutEvent">
                                <script>
                                        noMoreDetach = true;
                                </script>
                        </transition>
                        <transition event="error.interaction.detach" cond="(_event.data.error
== 'invalidstate')&&!noMoreDetach&&(_event.data.requestid == _data.reqid)">
                                <log expr="'G_G_GOT detach invalidstate error: ' + uneval(
_event.data )" />
                                <ixn:detach interactionid="_data.ixnid"
requestid="_data.reqid" />
                        </transition>
                        <transition event="error.interaction.detach" cond="(_event.data.error
== 'invalidstate')&&noMoreDetach&&(_event.data.requestid == _data.reqid)"
target="abnormal_exit">
                                <log expr="'G_G_GOT final detach invalidstate error: ' +
uneval( _event.data )" />
                        </transition>
                        <transition event="error.interaction.detach"
cond="(_event.data.error != 'invalidstate')&&(_event.data.requestid == _data.reqid)"
target="abnormal_exit">
                                <log expr="'G_G_GOT detach error: ' + uneval( _event.data )"
/>
                        </transition>
                        <transition event="interaction.detach.done"
cond="_event.data.requestid == _data.reqid" target="redirectcall">
                                <log expr="'G_G_GOT detach done: ' + uneval( _event.data )"/>
                        </transition>
                        <!--*** Targetless transition to intercept interaction.deleted event
***-->
                        <transition event="interaction.deleted"
cond="_event.data.interactionid == _data.ixnid">
                                <log expr="'G_G_GOT detached interaction deleted: ' + uneval(
_event.data )"/>
                                <log expr="'_genesys.ixn: ' + uneval( _genesys.ixn )" />
                        </transition>
                </state>
                <!--*********************************************************************-->
                <state id="redirectcall">
                        <onentry>
                                <ixn:redirect interactionid="_data.ixnid"
from="({'dn':'3001', 'switch':'tel_sw'})" to="({'dn':'7778', 'switch':'tel_sw_1'})"
requestid="_data.reqid" />
                        </onentry>
                        <transition event="error.interaction.redirect"
cond="_event.data.requestid == _data.reqid" target="abnormal_exit">
                                <log expr="'G_G_GOT redirect error: ' + uneval( _event.data
)" />
                        </transition>
                        <transition event="interaction.redirect.done"
cond="_event.data.requestid == _data.reqid" target="somedelay">
                                <log expr="'G_G_GOT redirect done: ' + uneval( _event.data
)"/>
                        </transition>
                </state>
                <!--*********************************************************************-->
                <!--*** Just to make old and new sessions coexist after ixn_detach
(additional detach verification) ***-->
                <state id="somedelay">
                        <onentry>
                                <send event="'SynchroEvent'" delay="'33s'"/>
                        </onentry>
                        <transition event="SynchroEvent" target="exit"/>
```

```
		</state>
		<!--*** Common event handler for primary ixn deletion (just to terminate this
test session) ***-->
		<transition event="interaction.deleted" cond="_event.data.interactionid ==
_data.ixnid" target="exit">
			<log expr="'G_G_GOT initial interaction deleted: ' + uneval(
_event.data )"/>
			<log expr="'_genesys.ixn: ' + uneval( _genesys.ixn )" />
		</transition>
	</state>
	<!--*******************************************************************-->
	<state id="abnormal_exit">
		<onentry>
			<log expr="'G_G_GOT ABNORMAL SESSION TERMINATION'" />
			<script>
				//Overwrite _data.test_result initial value that was set in
data block
				_data.test_result = 'FAILED';
			</script>
		</onentry>
		<transition target="exit"/>
	</state>
	<!--*******************************************************************-->
	<final id="exit">
		<onentry>
			<log expr="'G_G_GOT SESSION TERMINATION WITH TEST_RESULT = ' +
_data.test_result" />
		</onentry>
	</final>
</scxml>
```

Detect Consult Call

The following example illustrates the case when we are trying to detect whether the call that started the session is a `consult` call. The assumption is that the following SCXML file is configured on a Routing Point and the SCXML session is started when a call is made to the Routing Point.

Here are two scenarios:

1. Direct call to Routing Point
   - Customer makes a call and is connected to a Routing Point.
   - This is considered the primary call and the only active interaction.
   - All actions (`<queue:submit>`, `<dialog:playsound>`, etc) are applied to this interaction.
2. Consult call to Routing Point
   - Customer makes a call and is connected to an agent X.
   - This is considered the primary call and is not being monitored by Orchestration Server (the interaction is ownerless).
   - Agent X does a consult call to the Routing Point. This starts a SCXML session which is monitored by Orchestration.
   - The consult call is considered the `effective call` until the primary and consult calls are merged (which happens if agent X completes the transfer to the Routing Point). At that time, the consult call is no longer valid and the primary call is the `effective call`.
   - All actions (`<queue:submit>`, `<dialog:playsound>`, etc) are applied to the `effective call`.

Assumptions:

- At any time during the session, if the primary call is dead, the SCXML session will be terminated, regardless of the status of the consult call. This assumes the primary call and consult calls have not been merged.
- At any time during the session, if the effective call is dead, the SCXML session will be terminated.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:ixn="http://www.genesyslab.com/modules/interaction"
        initial="global">
        <script>
                var reqid;
                var consult_ixn_id;
                var primary_ixn_id;
                var effective_ixn_id;
                var sessionStarted = false;
        </script>
        <!--**************************************************************************-->
        <state id="global" initial="initial">
                <!--**************************************************************************-->
                <state id="initial">
                        <!--This ensures the session terminates after 10 minutes-->
                        <onentry>
                                <send event="'toExit'" delay="'600s'" />
                        </onentry>
                        <transition event="interaction.added" cond="sessionStarted == false">
                                <script>
                                        /* To avoid catching another 'interaction.added' event
                                        (caused by 'attach') in the same state again, set
sessionStarted to
                                        true. 'Attach' action could be done in a separate
state, but for
                                        the sake of simplicity and to minimize number of
states it is done
                                        here in initial state...*/
                                        sessionStarted = true;
                                        /* Assign interaction IDs that will be needed later
on ... */
                                        if(
_genesys.ixn.interactions[_event.data.interactionid].voice.type == 'consult' )
                                        {
                                                consult_ixn_id = _event.data.interactionid;
                                                primary_ixn_id =
_genesys.ixn.interactions[consult_ixn_id].parentid;
                                                effective_ixn_id = consult_ixn_id;
                                        }
                                        else
                                        {
                                                consult_ixn_id = undefined;
                                                primary_ixn_id = _event.data.interactionid;
                                                effective_ixn_id = primary_ixn_id;
                                        }
                                </script>
                                <log expr="'CONSULT_EXAMPLE: consult_ixn_id = ' +
consult_ixn_id" />
                                <log expr="'CONSULT_EXAMPLE: primary_ixn_id = ' +
primary_ixn_id" />
                                <log expr="'CONSULT_EXAMPLE: effective_ixn_id = ' +
effective_ixn_id" />
                                <if cond="consult_ixn_id ;!= undefined">
                                        <log expr="'CONSULT_EXAMPLE: Consult call started
strategy. Attaching primary call...'" />
                                        <ixn:attach requestid="reqid"
interactionid="primary_ixn_id" />
                                <else />
                                        <log expr="'CONSULT_EXAMPLE: Normal call started
strategy. Proceeding with session ...'" />
                                        <send event="'toProceed'" />
                                </if>
                        </transition>
                        <transition event="interaction.attach.done"
cond="_event.data.requestid == reqid" target="prewaiting_state" />
                        <!-- error.interaction.attach event (if happened) will be caught in
global state -->
                        <transition event="toProceed" target="CUSTOM_WORKING_STATE" />
                </state>
                <!--**********************************************************************-->
                <state id="prewaiting_state">
                        <onentry>
                                <!--This illustrates the case when the session is started by
a consult
                                        call (and that call is still alive here), sometimes
it makes sense
                                        to wait for some short amount of time. This time
could depend on
```

```
                                        how fast TServer completes transfer, or could be done
to avoid
                                        routing consult call during mute transfer, etc.-->
                                <log expr="'CONSULT_EXAMPLE: Continuing session with some
short delay...'" />
                                <send event="'toProceed'" delay="'1s'" />
                        </onentry>
                        <transition event="toProceed" target="CUSTOM_WORKING_STATE" />
                </state>
                <!--*********************************************************************-->
                <!--************ This is where your main logic goes ********************-->
                <!--*********************************************************************-->
                <state id="CUSTOM_WORKING_STATE" initial="route_to_agent">
                        <!--This will try to route the call to agent 703_sip. If it is not
                                successful within 3 seconds, it will transition to state
"dialog"
                                and play music. The attribute "clearontimeout" is set to
false so
                                router will continue trying to route to the agent while the
music is
                                playing.-->
                        <state id="route_to_agent">
                                <onentry>
                                        <queue:submit requestid="reqid"
interactionid="effective_ixn_id"
                                                priority="5" timeout="3"
clearontimeout="false">
                                                <queue:targets>
                                                        <queue:target type="agent"
name="'703_sip'" />
                                                </queue:targets>
                                        </queue:submit>
                                </onentry>
                                <transition event="error.queue.submit" target="dialog">
                                        <log expr="'ERROR WITH QUEUE SUBMIT: ' + uneval(
_event )" />
                                </transition>
                        </state>
                        <!-- This plays music for 60 seconds. -->
                        <state id="dialog">
                                <onentry>
                                        <dialog:playsound requestid="reqid"
interactionid="effective_ixn_id"
                                                type="'music'" resource="'music/on_hold'"
duration="60" />
                                </onentry>
                                <transition event="dialog.playsound.done.timeout" />
                                <transition event="dialog.playsound.done" target="exit" />
                                <transition event="error.dialog.playsound" target="error">
                                        <log expr="'ERROR PLAYING MUSIC: ' + uneval(_event)"
/>
                                </transition>
                        </state>
                        <transition event="queue.submit.done" target="exit">
                                <log expr="'QUEUE SUBMIT DONE.  Ending Session.'" />
                        </transition>
                        <transition event="interaction.partystatechanged"
cond="effective_ixn_id == _event.data.interactionid">
                                <log expr="'CONSULT_EXAMPLE: Got partystatechanged event: ' +
uneval(_event.data)" />
                        </transition>
                </state>
                <!--*********************************************************************-->
                <!--*********************************************************************-->
                <!--*********************************************************************-->
                <transition event="interaction.onmerge"
cond="_event.data.frominteractionid == consult_ixn_id && _event.data.tointeractionid ==
primary_ixn_id">
                        <script>
                                consult_ixn_id = undefined;
                                effective_ixn_id = primary_ixn_id;
                        </script>
                        <log expr="'CONSULT_EXAMPLE: Effective call ID changed because of
transfer completion: ' + uneval(_event)" />
                        <log expr="'CONSULT_EXAMPLE: consult_ixn_id = ' + consult_ixn_id" />
                        <log expr="'CONSULT_EXAMPLE: primary_ixn_id = ' + primary_ixn_id" />
                        <log expr="'CONSULT_EXAMPLE: effective_ixn_id = ' + effective_ixn_id"
/>
                </transition>
                <transition event="interaction.deleted"
                        cond="_event.data.interactionid == effective_ixn_id" target="exit">
                        <log expr="'CONSULT_EXAMPLE: Effective call is dead. Exiting...: ' +
uneval(_event)" />
                </transition>
                <transition event="interaction.deleted"
                        cond="_event.data.interactionid == primary_ixn_id &&
consult_ixn_id != undefined"
                        target="exit">
                        <log expr="'CONSULT_EXAMPLE: Primary call is dead, consult call is
alive and useless. Exiting...: ' + uneval(_event)" />
                </transition>
                <!--In case none of the other events are triggered, this will end the
                        session after number of minutes specified at the strategy beginning-->
                <transition event="toExit" target="exit">
                        <log expr="'CONSULT_EXAMPLE: Possibly stuck session is self-
destructing. Exiting...: ' + uneval(_event)" />
                </transition>
                <!--This will catch all the errors that are not processed elsewhere-->
                <transition event="error.*" target="error">
                        <log expr="'CONSULT_EXAMPLE: ERROR AT GLOBAL LEVEL'" />
                        <log expr="'CONSULT_EXAMPLE: Got error event: ' + uneval( _event )" />
                </transition>
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

- When agent X initiates a transfer or consult to the Routing Point, it will trigger a SCXML session to be created and will wait for the `interaction.added` event.
- After the `interaction.added` event is received, it will set the `consult_ixn_id`, `primary_ixn_id`, and `effective_ixn_id` depending on whether the session was started by a regular call, or a consult call to the Route Point.
- If the SCXML application detects that the call from Agent X to the Routing Point is of type `consult`, we attach the parent interaction (the primary call which is ownerless) to the current session (see interaction attach for more details about ownership).
- The `interaction.attach.done` event will trigger a transition to the `prewaiting_state`, where we put in a delay. This delay is needed depending on how fast TServer completes the transfer, or is sometimes done to avoid routing a consult call during a mute transfer.
- The `CUSTOM_WORKING_STATE` is where you would put your main logic. In this example, we first try to route the call to agent `703_sip`. If this is not successful within 3 seconds, we transition to the `dialog` state and play music for 60 seconds.

- At any time during the session, if agent X decides to complete the transfer to the Routing Point or to agent Y (if the consult call was routed from the Routing Point to agent Y), the primary and consult calls are merged, and the event `interaction.onmerge` is raised. This event triggers a transition in the SCXML application and redefines the variables `consult_ixn_id`, and `effective_ixn_id` since the consult interaction is deleted during the merge. The `consult_ixn_id` will no longer be valid and is set to `undefined`. The `effective_ixn_id` is changed from the consult call to the primary call and should be used from this point forward for all functions and actions that require an interaction ID.
- Exiting the session is triggered by any of the following situations:
    - The call is successfully routed to agent `703_sip`.
    - Music has been played for 60 seconds.
    - There was a problem playing the file `music/on_hold`.
    - The effective call is deleted (effective call is the consult call until the consult or transfer is complete, at which time, it is the only call left).
    - The primary call is deleted before the consult or transfer is complete (the consult call can still be alive but is useless at this point).
    - Any `error.*` events that are raised during the session.
    - The session may be stuck and self-destucts 10 minutes after it was created.

Detect User Data Changes

The following SCXML strategy shows how to detect changes in user data for an interaction.

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        initial="waitForInteraction">
        <script>
                var userdata = ({});
                function detectChanges( oldObj, newObj, objDiffs )
                {
                        if ( null == oldObj || null == newObj ||
                                "undefined" ==        typeof( oldObj ) || "undefined" ==
typeof( newObj ) )
                        {
                                __Log( 'Invalid        userdata' )
                                return false;
                        }

                        var key;
                        for ( key in oldObj )
                        {
                                if ( newObj.hasOwnProperty( key ) )
                                {
                                        if ( oldObj[key] !=        newObj[key] )
                                        {
                                                objDiffs.push({ userdata : "updated" ,
key:        key, oldvalue :        oldObj[key], newvalue :        newObj[key] });
                                        }
                                }
                                else
                                {
                                        objDiffs.push({ userdata: "deleted", key : key,
value :        oldObj[key] });
                                }
                        }
                        for ( key in newObj )
                        {
                                if ( !oldObj.hasOwnProperty( key ) )
                                {
                                        objDiffs.push({ userdata: "added", key : key, value :
newObj[key] });
                                }
                        }

                        return true;
                }
        </script>
        <datamodel>
                <data id="reqid" />
        </datamodel>
        <state id="waitForInteraction">
                <transition event="interaction.added" target="globalstate">
                        <script>
                                _data.ixnid = _event.data.interactionid;
                        </script>
                </transition>
        </state>
        <state id="globalstate">
                <onentry>
```

```
                        <script>
                                userdata = _genesys.ixn.interactions[_data.ixnid].udata;
                                __Log('Userdata initial value = ' + uneval( userdata ) );
                        </script>
                </onentry>
                <initial>
                        <transition target="addUdata" />
                </initial>
                <transition event="interaction.udata.changed"
                        cond="_genesys.ixn.interactions[_data.ixnid].udata.hasOwnProperty(
'timeToExit' )"
                        target="exit" />
                <transition event="interaction.udata.changed">
                        <script>
                                var objDiffs =[];
                                if ( detectChanges( userdata,
_genesys.ixn.interactions[_data.ixnid].udata, objDiffs ) )
                                {
                                        __Log( uneval( objDiffs ) );
                                }
                                __Log('Userdata updated to = ' + uneval(
                                _genesys.ixn.interactions[_data.ixnid].udata ) )
                                userdata =
                                _genesys.ixn.interactions[_data.ixnid].udata;
          </script>
                </transition>
                <state id="addUdata">
                        <onentry>
                                <script>
                                        var myData = { name : "Smith, John", age : 45 };
                                        _genesys.ixn.setuData( myData );
                                </script>
                        </onentry>
                        <transition target="updateUdata" />
                </state>
                <state id="updateUdata">
                        <onentry>
                                <script>
                                        _genesys.ixn.setuData( { age : 25 } );
                                </script>
                        </onentry>
                        <transition target="deleteUdata" />
                </state>
                <state id="deleteUdata">
                        <onentry>
                                <script>
                                        _genesys.ixn.deleteuData( 'name' );
                                </script>
                        </onentry>
                        <transition target="endInteraction" />
                </state>
                <state id="endInteraction">
                        <onentry>
                                <script>
                                        var exitFlag = { timeToExit : "Y" };
                                        _genesys.ixn.setuData( exitFlag );
                                </script>
                        </onentry>
                </state>
        </state>
        <final id="exit" />
        <final id="error" />
</scxml>
```

In this example, the "detectChanges" function is called **five** times (see sample ORS log below). It is first called when ORS information such as ORDbid, ORSession, and ORUrl are added to the interaction user data. It is called a second time after "name" and "age" are added to the user data during state "addUdata". It is called a third time after the "age" is modified during the state "updateUdata". It is called a fourth time after the "name" is deleted during the state "deleteUdata". It is called a fifth time after "timeToExit" is added to the user data during state "endInteraction", which signals the strategy to end when it transitions to the state "exit".

The inputs to the function "detectChanges" are the userdata object before the "interaction.udata.changed" event, and the userdata object after the "interaction.udata.changed" event. The output is the "objDiffs" object which keeps track of changes such as additions, deletions, and modifications.

```
METRIC <transition sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' name='globalstate'
event='interaction.udata.changed' line='64' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='[{userdata:"added", key:"ORDbid",
value:"143"},
       {userdata:"added", key:"ORSession", value:"2M4DTARKVT2MBFSKSCVVHVM08K000001"},
       {userdata:"added", key:"ORUrl", value:"http://localhost:17011/scxml"}]' label=''
level='1' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='Userdata updated to =
({ORDbid:"143", ORSession:"2M4DTARKVT2MBFSKSCVVHVM08K000001",
       ORUrl:"http://localhost:17011/scxml"})' label='' level='1' />
...
METRIC <event_queued sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' name='interaction.udata.changed'
type='external' />
METRIC <event_processed sid='2M4DTARKVT2MBFSKSCVVHVM08K000001'
name='interaction.udata.changed' disposition='transition selected' />
METRIC <transition sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' name='globalstate'
event='interaction.udata.changed' line='64' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='[{userdata:"added", key:"age",
value:45},
       {userdata:"added", key:"name", value:"Smith, John"}]' label='' level='1' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='Userdata updated to =
({ORDbid:"143", ORSession:"2M4DTARKVT2MBFSKSCVVHVM08K000001",
       ORUrl:"http://localhost:17011/scxml", age:45, name:"Smith, John"})' label='' level='1'
/>
...
METRIC <event_queued sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' name='interaction.udata.changed'
type='external' />
METRIC <event_processed sid='2M4DTARKVT2MBFSKSCVVHVM08K000001'
name='interaction.udata.changed' disposition='transition selected' />
METRIC <transition sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' name='globalstate'
event='interaction.udata.changed' line='64' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='[{userdata:"updated", key:"age",
oldvalue:45, newvalue:25}]' label='' level='1' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='Userdata updated to =
({ORDbid:"143", ORSession:"2M4DTARKVT2MBFSKSCVVHVM08K000001",
       ORUrl:"http://localhost:17011/scxml", age:25, name:"Smith, John"})' label='' level='1'
/>
...
METRIC <event_queued sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' name='interaction.udata.changed'
type='external' />
METRIC <event_processed sid='2M4DTARKVT2MBFSKSCVVHVM08K000001'
name='interaction.udata.changed' disposition='transition selected' />
METRIC <transition sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' name='globalstate'
event='interaction.udata.changed' line='64' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='[{userdata:"deleted", key:"name",
value:"Smith, John"}]' label='' level='1' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='Userdata updated to =
({ORDbid:"143", ORSession:"2M4DTARKVT2MBFSKSCVVHVM08K000001",
       ORUrl:"http://localhost:17011/scxml", age:25})' label='' level='1' />
...
METRIC <event_queued sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' name='interaction.udata.changed'
type='external' />
METRIC <event_processed sid='2M4DTARKVT2MBFSKSCVVHVM08K000001'
name='interaction.udata.changed' disposition='transition selected' />
METRIC <transition sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' name='globalstate'
event='interaction.udata.changed' line='64' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='[{userdata:"added",
key:"timeToExit", value:"Y"}]' label='' level='1' />
METRIC <log sid='2M4DTARKVT2MBFSKSCVVHVM08K000001' expr='Userdata updated to =
({ORDbid:"143", ORSession:"2M4DTARKVT2MBFSKSCVVHVM08K000001",
       ORUrl:"http://localhost:17011/scxml", age:25, timeToExit:"Y"})' label='' level='1' />
```

Route to Fetched Targets With Invoking SCXML Strategies

The following SCXML strategy uses the results of a fetch request to construct routing targets.

Fetched files content

- File //myhost/Fetch_1.txt:

  ```
  {"id":813, "id1":819}
  ```

- File //myhost/Fetch_2.txt:

  ```
  {"id":511, "id1":517}
  ```

Root strategy

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        xmlns:dialog="www.genesyslab.com/modules/dialog"
        xmlns:session="www.genesyslab.com/modules/session"
        xmlns:xi= "http://www.w3.org/2001/XInclude"
        initial="waitinteraction">
<script>
    var data1;
    var data2;
</script>
<state id="waitinteraction">
   <transition event="interaction.added" target="getdata1"/>
</state>
<state id="getdata1">
    <onentry>
       <session:fetch srcexpr="'http://myhost/Fetch_1.txt'"/>
    </onentry>
    <transition event="session.fetch.done" target="getdata2">
        <script>
           data1= JSON.parse(_event.data.content);
        </script>
    </transition>
    <transition event="error.session.fetch" target="error"/>
</state>
<state id="getdata2">
    <onentry>
       <session:fetch srcexpr="'http://myhost/Fetch_2.txt'"/>
    </onentry>
    <transition event="session.fetch.done" target="initial">
        <script>
           data2= JSON.parse(_event.data.content);
        </script>
```

```
    </transition>
    <transition event="error.session.fetch" target="error"/>
</state>
<state id="initial">
 <transition cond="_genesys.ixn.interactions[0].userdata['switch']=='1'"
 target="queued"/>
 <transition cond="_genesys.ixn.interactions[0].userdata['switch']=='2'"
 target="queued1"/>
</state>
<state id="queued">
 <onentry>
  <queue:submit queue="'vq1'" priority="5" timeout="2">
   <queue:targets type="skill" statserver="'Single_StatServer'">
    <queue:target
        skillexpr="'switch=1&id>' + data1.id + '&id<' + data1.id1"/>
   </queue:targets>
  </queue:submit>
 </onentry>
 <transition event="queue.submit.done" target="exit" />
 <transition event="error.queue.submit" target="q2.queued" />
</state>
<state id="queued1">
 <onentry>
  <queue:submit queue="'vq1'" priority="5" timeout="2">
   <queue:targets type="skill" statserver="'Single_StatServer'">
    <queue:target
         skillexpr="'switch=1&id>'+ data2.id + '&id<' + data2.id1"/>
   </queue:targets>
  </queue:submit>
 </onentry>
 <transition event="queue.submit.done" target="exit" />
 <transition event="error.queue.submit" target="q3.queued" />
</state>
<xi:include  resolved ="q2" href="Inv_Queue_2.xml" xpointer="queued" >
<xi:include  resolved ="q3" href="Inv_Queue_3.xml" xpointer="queued" >
<final id = "exit"/>
<final id = "error"/>
</scxml>
```

Strategies making second-chance routing to queues (if first queue:submit failed)

- File //myhost/Inv_Queue_2.xml:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        initial="initial">
<state id="initial" >
  <transition event="interaction.added" target="queued"/>
</state>
<state id="queued">
  <onentry>
    <queue:submit queue="'vq1'" priority="5" timeout="100">
      <queue:targets type="queue">
        <queue:target name="'8112_sw1'"/>
      </queue:targets>
    </queue:submit>
  </onentry>
  <transition event="queue.submit.done" target="exit" />
  <transition event="error.queue.submit" target="error" />
</state>
<final id = "exit"/>
<final id = "error"/>
</scxml>
```

- File //myhost/Inv_Queue_3.xml:

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
        xmlns:queue="www.genesyslab.com/modules/queue"
        initial="initial">
<state id="initial" >
  <transition event="interaction.added" target="queued"/>
</state>
<state id="queued">
  <onentry>
    <queue:submit queue="'vq1'" priority="5" timeout="100">
      <queue:targets type="queue">
        <queue:target name="'8113_sw1'"/>
      </queue:targets>
    </queue:submit>
  </onentry>
  <transition event="queue.submit.done" target="exit" />
  <transition event="error.queue.submit" target="error" />
</state>
<final id = "exit"/>
<final id = "error"/>
</scxml>
```

Orchestration Server Troubleshooting

Common Problems

Issue	What to do
Incorrect processing of call in following scenario:	Issue: 1. An inbound call with certain DID (DNIS) comes from Media Gateway to SIP Server and directly forwarded to GVP. 2. GVP is represented by a Trunk DN in SIP Server, meaning SIP Server won't generate DN related TEvent on it. 3. After caller's interaction with GVP is finished, GVP transfers the call to a Routing Point at SIP Server 4. Time of call processing in GVP is higher than ~30 sec. Resolution: 1. Configure a TrunkGroup DN that points to GVP, instead of the Trunk DN. This will result in generating Tevents on the Trunk Group, which will ensure proper completion of call create transaction in this scenario. 2. Increase the timeout to the length of time to wait for the event before failing the call create transaction. By default it's 30 seconds. If in the scenario described here, the call spends more than 30 seconds on GVP, call create transaction would fail. To increase the timeout, set option "cti-transaction-timeout" in section "gts" in the ORS application to the desired value in seconds. This option can have a maximum value of 600 (10 minutes).
Unexpected crash after some period of time ORS stops creating sessions	The most common cause for this problem is that sessions are not ending. In Orchestration, a session ends by transition to a <final> element. Orchestration sessions can be started in a number of ways and have a number of purposes. For sessions that are based around voice interactions, it should be made certain that the session ends. This is typically done by detecting the event interaction.deleted and then transitioning to a <final>. For projects built with Composer, this can be easily done by adding an exception handler to

Issue	What to do
	the default Entry point.
Internal communication delays	In cases where there are unexpected delays within the SCXML application execution, the cause may be unneccessary attempts to resolve 'localhost' through DNS. Internal thread communication employs this hostname. Check the /etc/hosts file to determine if localhost is defined as 127.0.0.1, for example for ipv4.
ORS occasionally fails to fetch document with error "*Recv failure: Connection reset by peer*" if idle time between calls exceeds web server Connection Timeout	This was observed on a RedHat installation of ORS where it was discovered that TCP_KEEPALIVE was turned off on ORS sockets. ORS attempts to re-use existing connections but when TCP_KEEPALIVE is disabled, libcurl fails to detect when TCP connections are disconnected. TCP_KEEPALIVE can be forcibly enabled by loading libkeepalive.so via LD_PRELOAD (add path of libkeepalive.so to LD_PRELOAD environment variable) prior to starting ORS.

ECMAScript

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 >

TTTTFor <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.

Name	Required	Type	Default Value	Valid Values	Description
_desc	false	string	none	Any valid string	This allows the 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> (Since ORS 8.1.200.40, SCXML 8.1.000.77)

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 child element <content> of <invoke> is *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

Name	Required	Type	Default Value	Valid Values	Description
					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 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.

Name	Required	Type	Default Value	Valid Values	Description
					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.
_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

Name	Required	Type	Default Value	Valid Values	Description
					_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".
_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 (for example, 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

Name	Required	Type	Default Value	Valid Values	Description
					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.
_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

Name	Required	Type	Default Value	Valid Values	Description
					Primary.
_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 order. Transitions of deepest scope (most nested) are executed first. Ties for scope are broken

Name	Required	Type	Default Value	Valid Values	Description
					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
_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

Name	Required	Type	Default Value	Valid Values	Description
					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 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

Name	Required	Type	Default Value	Valid Values	Description
					of this state (before the <onentry> elements are executed). This is used to guarantee 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 how meaningful it

Name	Required	Type	Default Value	Valid Values	Description
					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 [1]. 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 [2], 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>

Name	Required	Type	Default Value	Valid Values	Description

					For example:
					
					`<xi:include 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 extension

Name	Required	Type	Default Value	Valid Values	Description
					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 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

Name	Required	Type	Default Value	Valid Values	Description
					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.org/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 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*.