Orchestration Server Developer's Guide

Orchestration Getting Started Guide

4/28/2025

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.