Orchestration Server Developer's Guide

Core Extensions

12/12/2025

Contents

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`		Not supported.	Not supported.
POST, PUT		Content of the <content> element will be URL-encoded without prior evaluation. This element cannot be used together with <param>.	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.