Developer's Guide

Customizing the Engagement Strategy

4/22/2025

Customizing the Engagement Strategy

Contents

When you create your Web Engagement application, Genesys Web Engagement also creates default Engagement Logic and Chat Routing SCXML strategies in the **\apps*application_name*_composer-project** folder. Orchestration Server (ORS) uses these strategies to decide whether and when to make a proactive offer and which channels to offer (chat or web callback).

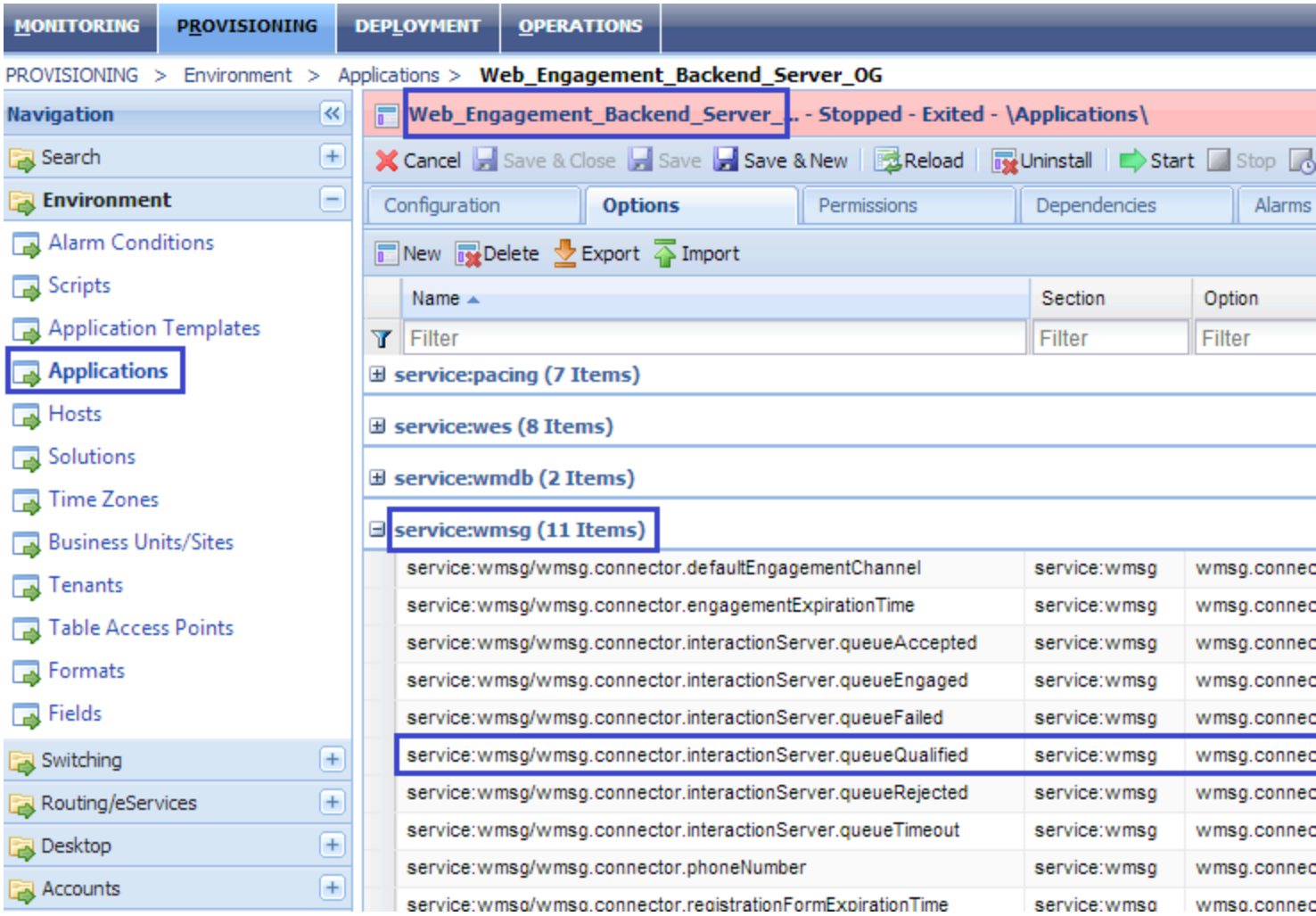
The Engagement Logic strategy processes Genesys Web Engagement interactions, and consists of sub-workflows to handle: general processing, decision making, obtaining additional information from the Cassandra database through the REST API, and contacting the Backend Server with instructions according to the engagement (or non-engagement) process.

You can modify the Engagement Logic SCXML by importing the Composer project into Composer. The project is located here: **\apps*application name*_composer-project\WebEngagement_EngagementLogic**. Refer to the sections below for details about the Engagement Logic strategy and how it can be modified.

Note: The strategies deployed out of the box with GWE 8.1.2 are made for Composer 8.1.300.89 projects. If you use a newer version of Composer, make sure that you upgrade these projects by following the upgrade procedure.

Main Interaction Process and Workflow

The default entry point to the Engagement Logic strategy is the Interaction Queue specified in the wmsg.connector.interactionServer.queueQualified option on the Backend Server application.



The Interaction Queue

Passing Parameters into the Engagement Logic Strategy

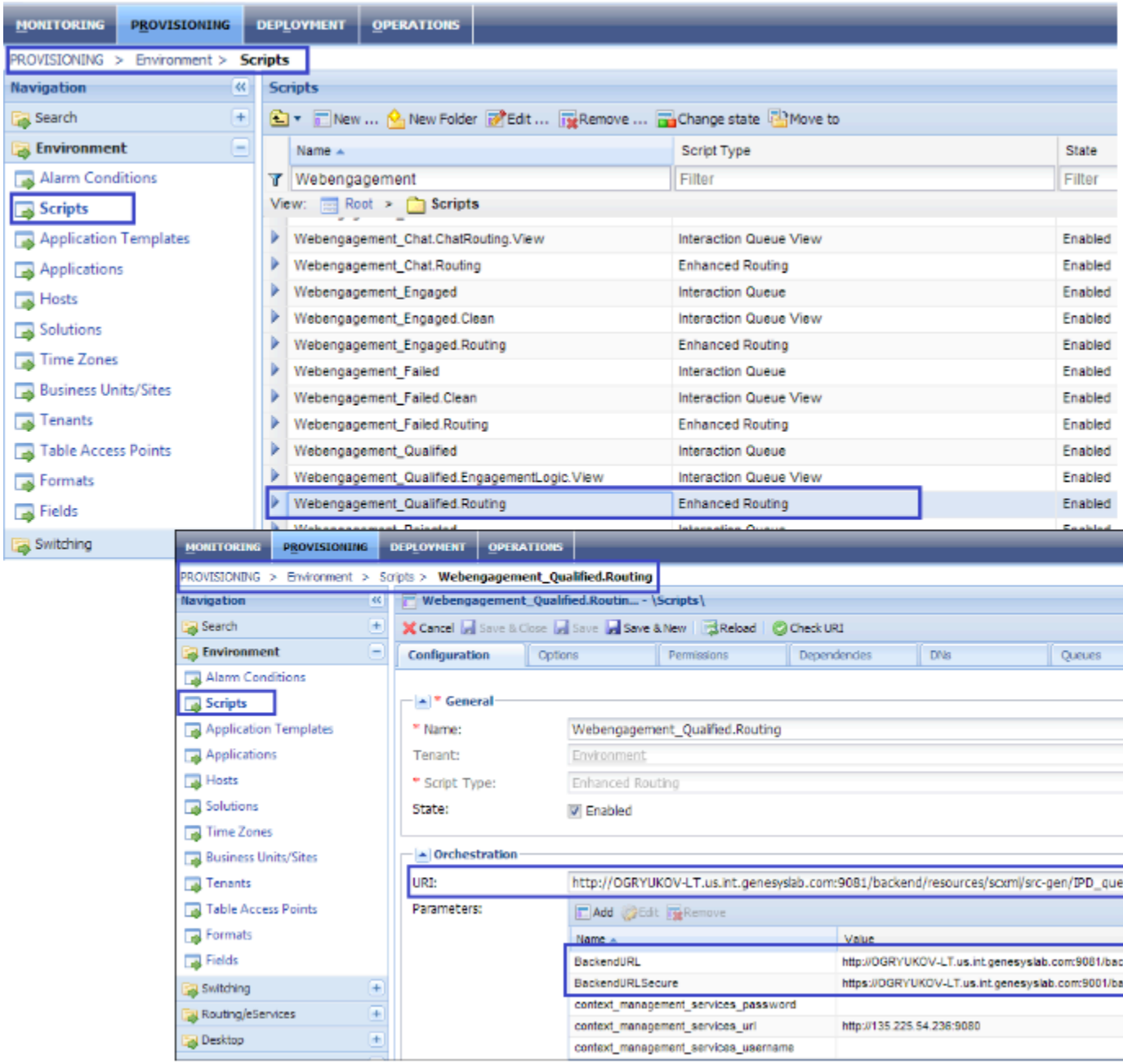
When Genesys Web Engagement creates an engagement attempt, the Backend Server creates an Open Media interaction of type **webengagement** and places it into the Interaction Queue specified by the **wmsg.connector.interactionServer.queueQualified** option. By default, this option is set to the Webengagement_Qualified queue. Orchestration Server (ORS) monitors this queue and pulls the interaction to process it with the Engagement Logic strategy.

Since ORS does not connect to the Backend Server(s), certain parameters must be passed to the Engagement Logic strategy in order to provide ORS with the data it needs.

1. The address where the SCXML strategy is located. **Note:** The default Engagement Logic and Chat Routing strategies are located as resources under the Backend Server. Provisioning automatically specifies this address in the related Configuration Server objects when GWE is installed. Since you can host strategies in other places, you can manually update the parameters in the related objects.
2. The address where the Backend Server can be accessed (if a secure address is present, pass this as well). This information is used to issue REST requests to the GWE Cassandra database and to start or

cancel the engagement procedure through the Backend Server.

The parameters are passed to ORS through the Routing script object Webengagement_Qualified.Routing that is associated with the Webengagement_Qualified Interaction Queue.



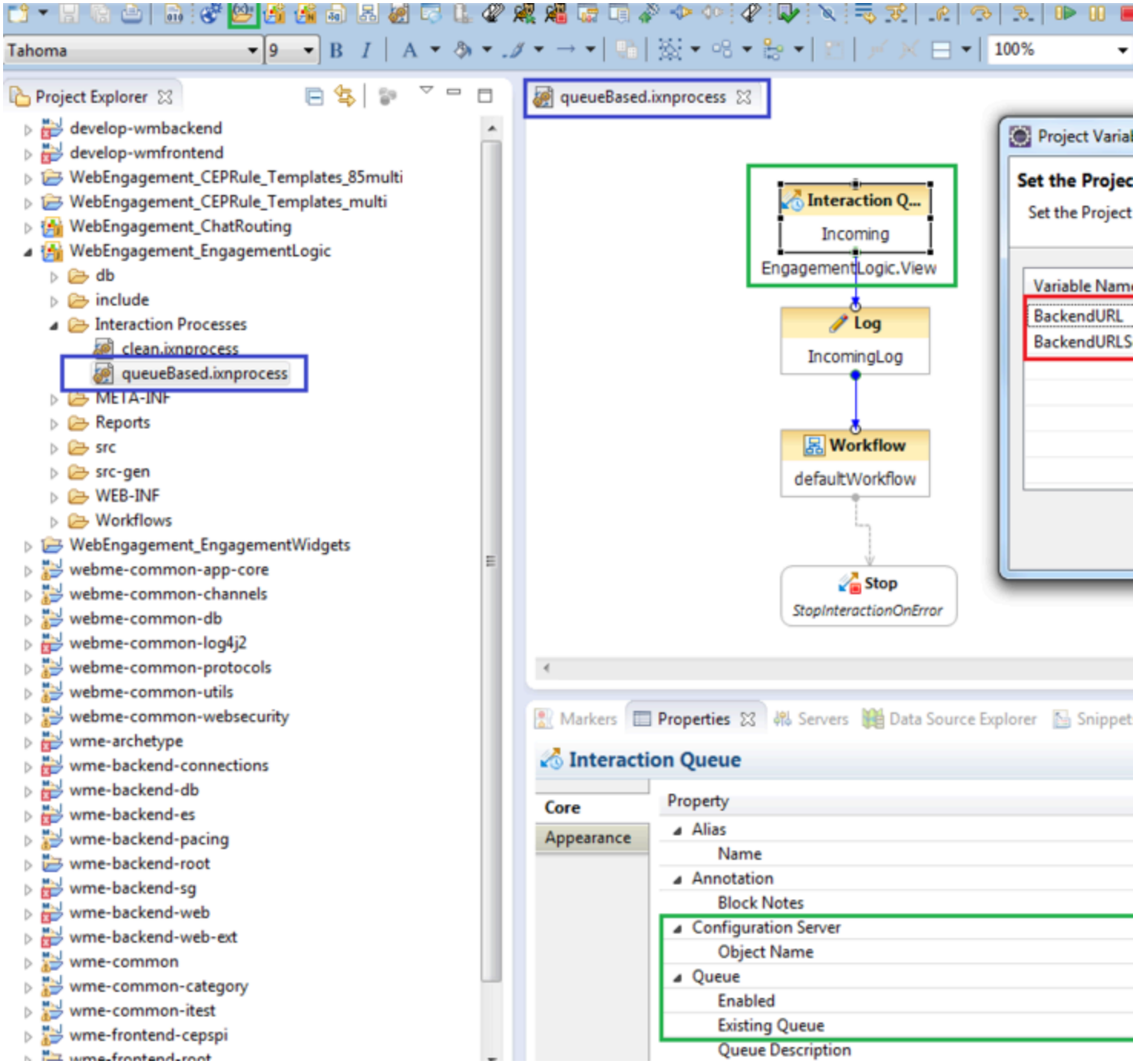
The Webengagement_Qualified.Routing

The Engagement Logic strategy has two interaction processes:

- **clean.ixnprocess** — This process is explained in Cleaning Interaction Process
- **queueBased.ixnprocess** — This process features the major logic for the strategy.

To access BackendURL and BackendURLSecure (as shown in the above image), use the button marked with a green square, which opens Project Variables. **Note:** In order to access Project Variables, your current tab in Composer must display Interaction Process (not Workflow).

The block and properties related to the entry point Interaction Queue (Webengagement_Qualified) are marked with green rectangles shown in the following image:



The **queueBased.ixnprocess**

After the interaction is taken into processing, it is placed into a set of workflows for processing. All workflows have notes related to specific blocks, however, this document highlights the most important items.

Preventing Interaction Termination into Sub-flows

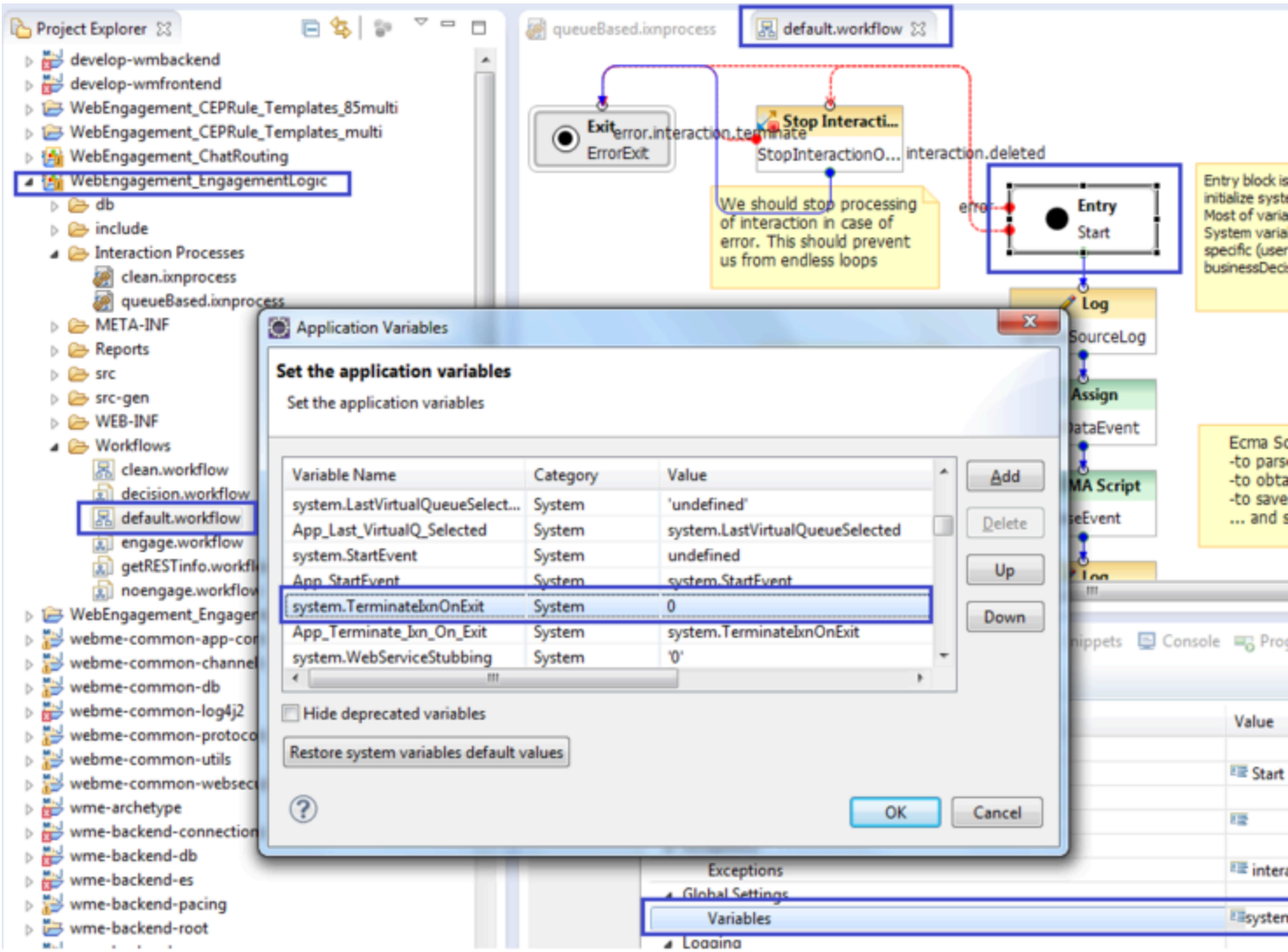
For all workflows, you must make sure that the workflow is configured to **not** terminate the interaction upon exiting. If this step is not followed, the entire interaction process will not be able to finish due to termination of the interaction in one of the sub-flows.

Note: Out-of-the-box Engagement Logic strategies already have the correct specified value (`0`) for the **system.TerminateIxnOnExit** variable.

You must perform the following steps to turn-off the termination of the interaction at the end of the sub-flow:

1. Open the workflow diagram in Composer (note that in the images, it is shown as **default.workflow**).
2. Select the **Entry** block, and in the properties of this block, open **Global Settings > Variables**.
3. Scroll down and locate the variable **system.TerminateIxnOnExit**. Set the value to `0`.

See the process in the following image:

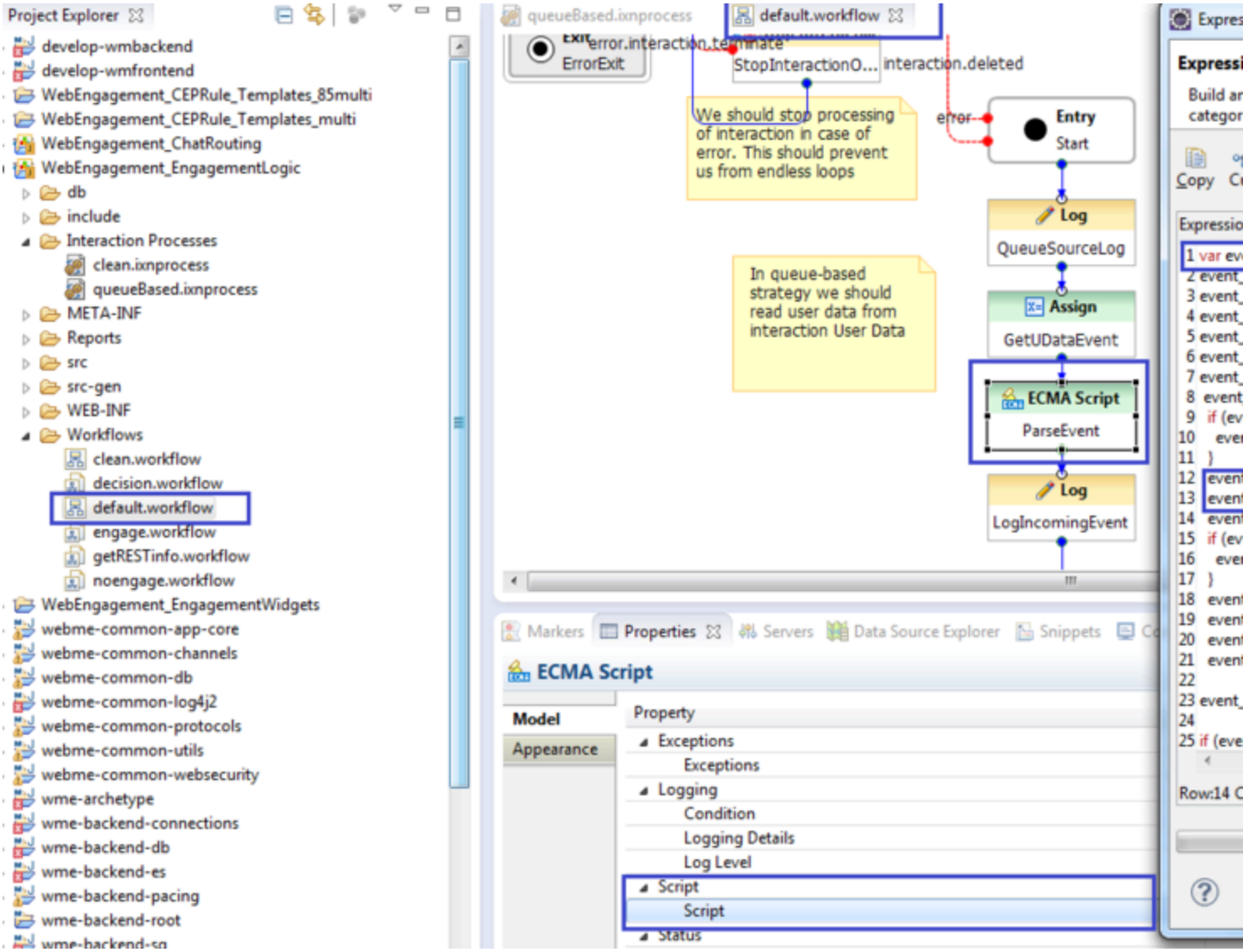


Turn off interaction termination

Parsing User Data from the webengagement Interaction and Passing it into Sub-flows

One of the most important items for Engagement Logic is the ability to access User Data of webengagement interaction. This data is fulfilled inside of the Backend Server and includes, among other items, information provided by a pacing algorithm. The following is the process of passing basic pacing algorithm data into a decision sub-flow.

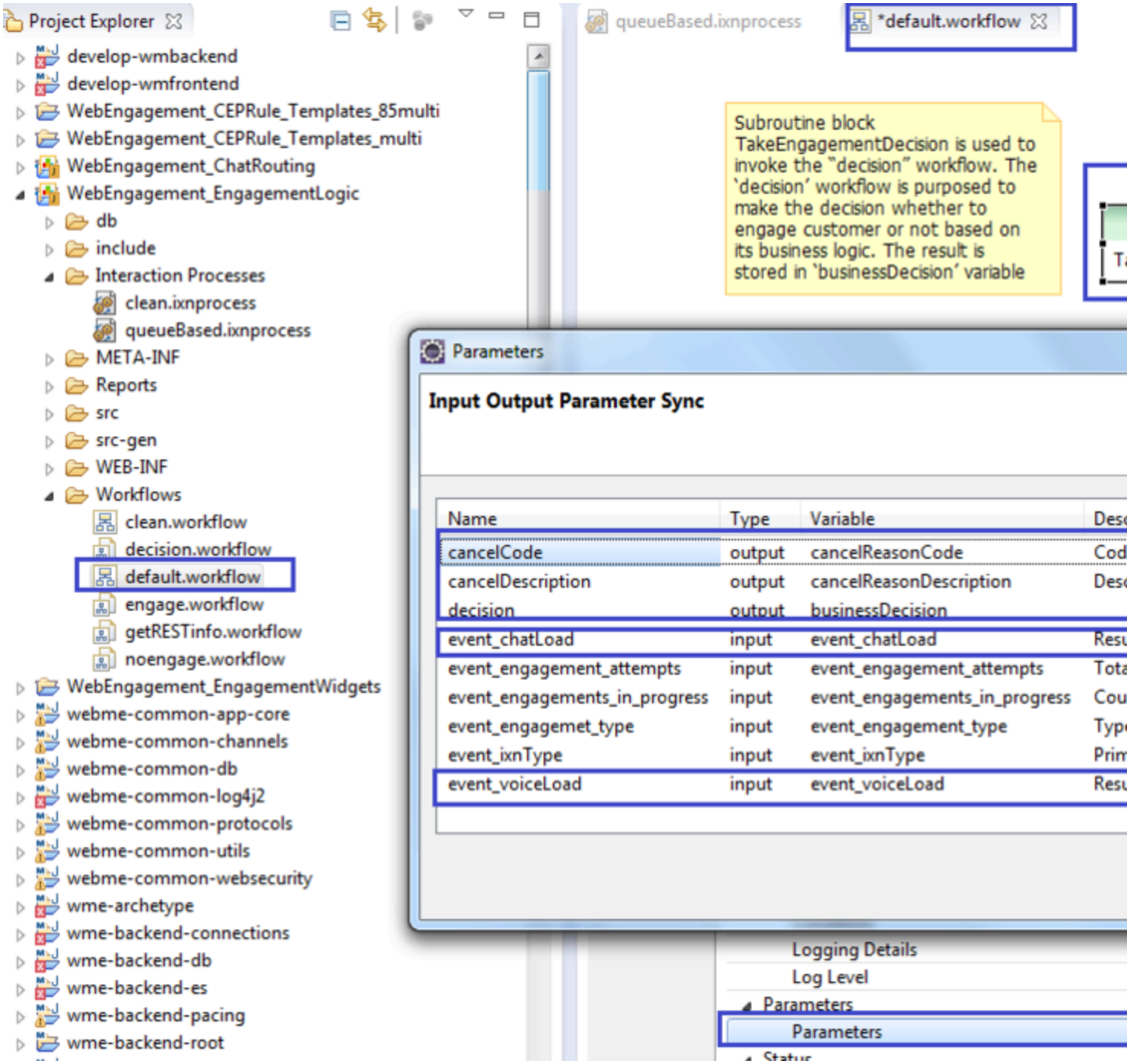
The first step is to parse User Data available in the webengagement interaction. This is done in the **ParseEvent** block:



The ParseEvent block

The variable **jsonEvent**, which is present in the described block is created in the **GetUDataEvent** block before as the following:
_genesys.ixn.interactions[system.InteractionID].udata.jsonEvent that is, from the current interaction we take key **jsonEvent** from User Data.

After data is parsed and assigned to variables, it can be propagated to sub-flows and used there. Sub-flows are also able to pass output data in a backward direction. In the example we pass (among other) parameters **event_chatLoad** and **event_voiceLoad** into **decision.workflow** and obtain back parameters **cancelCode**, **cancelDescription** and **decision**:



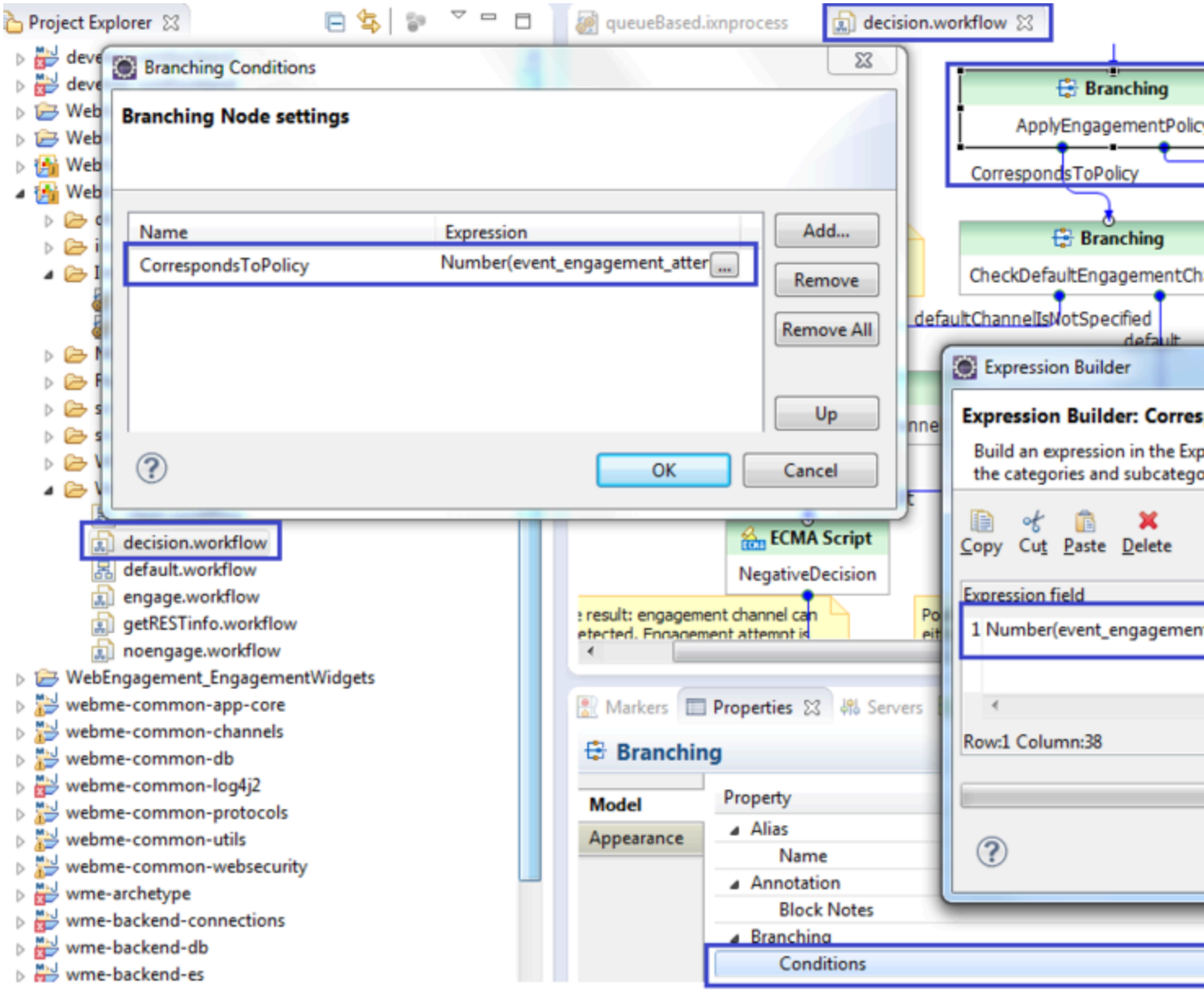
Passing output data

Engagement Policy (Decision Workflow)

Engagement policy is the other name of decision workflow.

Consider the most important points provided by the out-of-the box strategy:

Check count of engagement attempts already proposed to the current visitor. This check is executed in the **ApplyEngagementPolicy** block (see image below). Default value is 3, which means that no more that three engagement attempts should be proposed for a particular visitor. **Note:** If the engagement attempt was closed by a timeout, it will not be taken into count, as soon as there is no guarantee if the visitor has seen it at all. For example, the invitation may appear on a non-active browser tab or window.

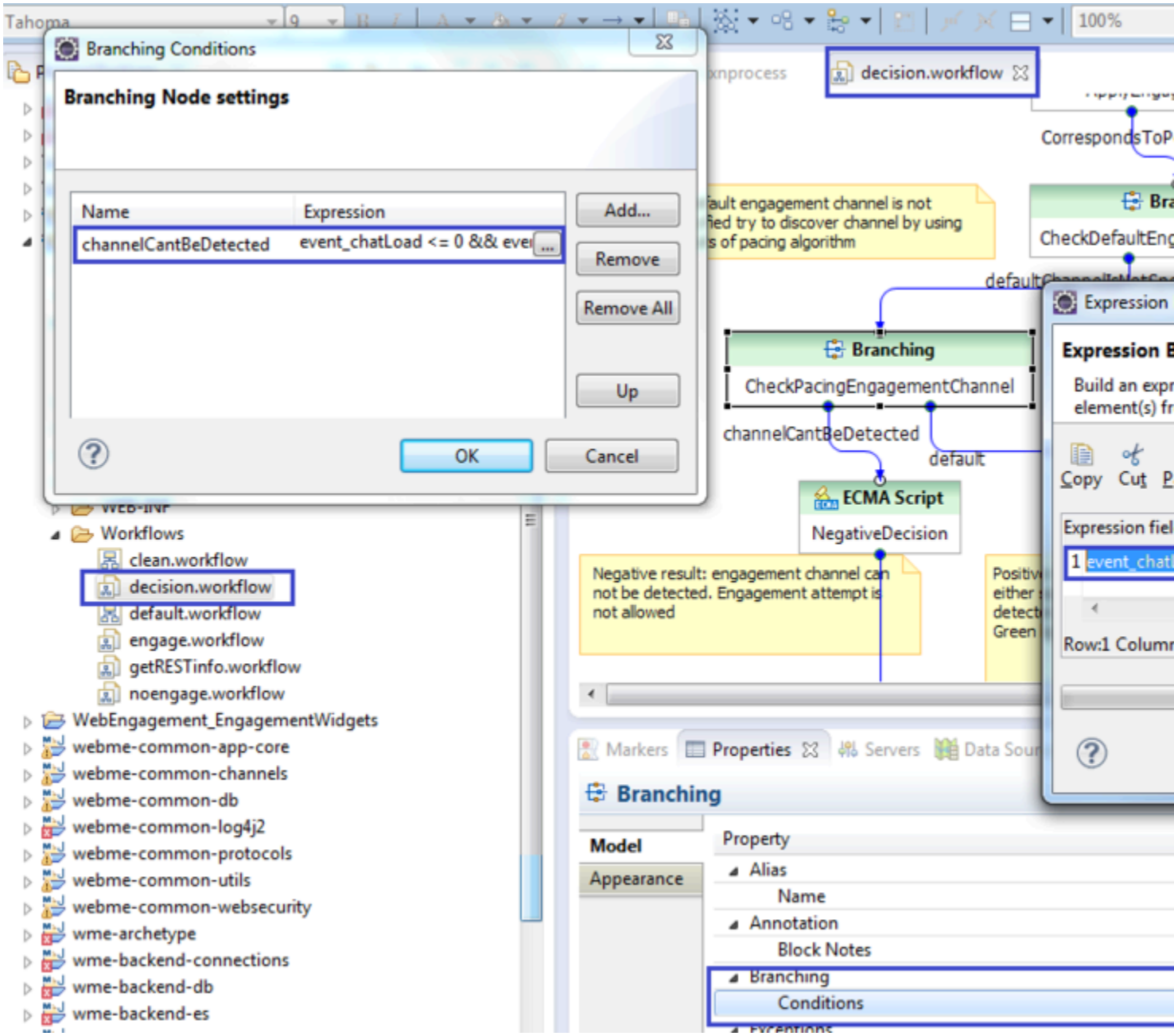


Check the engagement attempts count.

Check pacing information. This is executed inside of the **CheckPacingEngagementChannel** block.

Note: Out-of-the-box strategy operates only general information obtained from pacing algorithm: variables **event_chatLoad** and **event_voiceLoad**, passed from **default.workflow**, contain accumulated (by channel) count of interactions that can be triggered in the particular moment. It is possible to pass into the decision workflow detailed information provided by the pacing algorithm and build a more sophisticated decision maker. The image below shows the general idea: do **not** engage the visitor if the count of available "interactions to produce"

is 0 for both channels:



Check the pacing information.

Obtaining Data from the GWE Cassandra Database through REST Requests

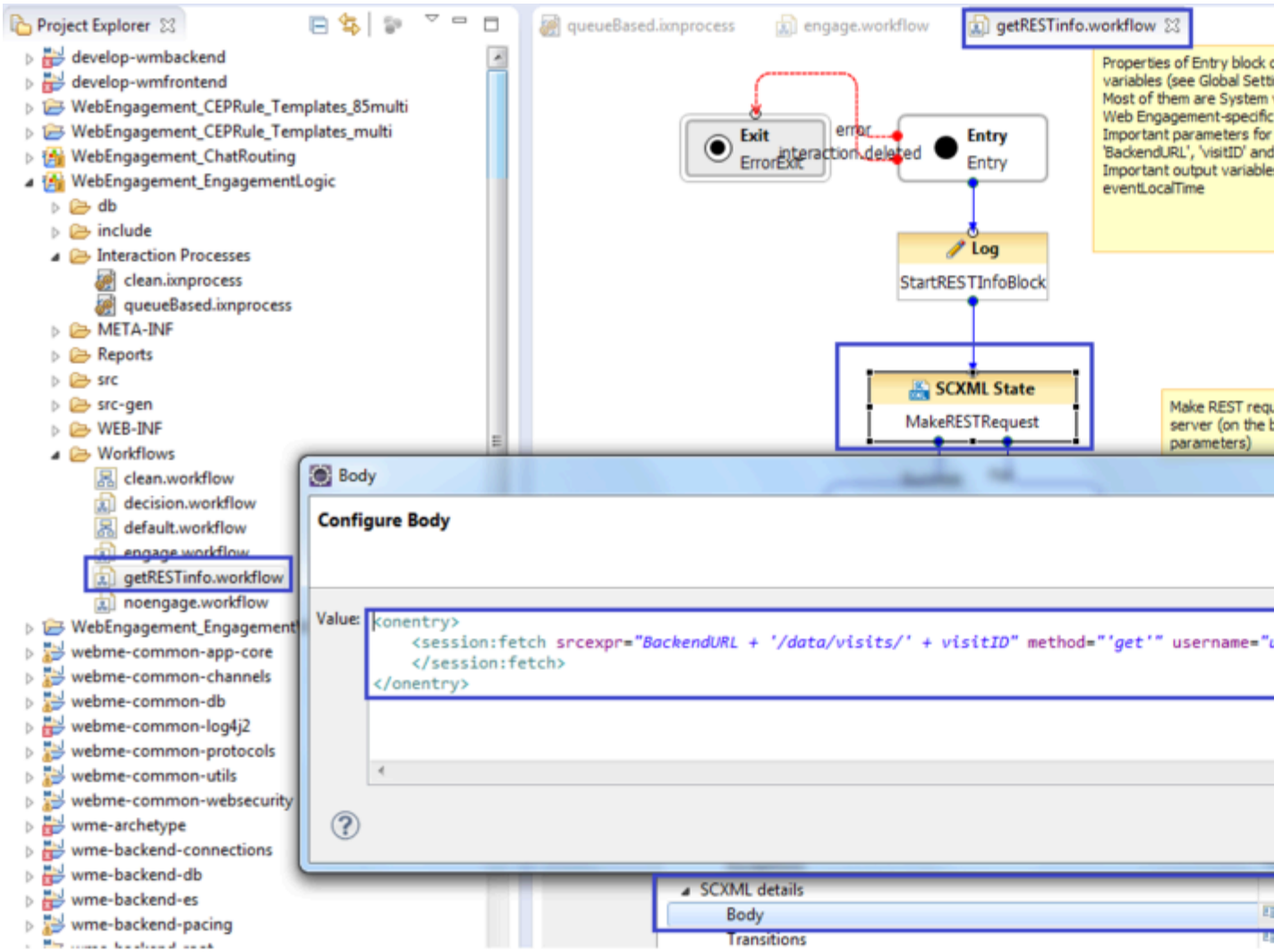
Requesting data from Backend Server through the REST

During the decision making process, it might be useful to access data from the Web Engagement Cassandra database. For example, to check additional parameters that are collected there.

The out-of-the-box Engagement Strategy provides an example of accessing the Cassandra database in order to get the **TimezoneOffset** of the visitor's browser, and correspondingly modify the greetings *good evening*, *good morning*, and so on.

Consider how Engagement Strategy does this task.

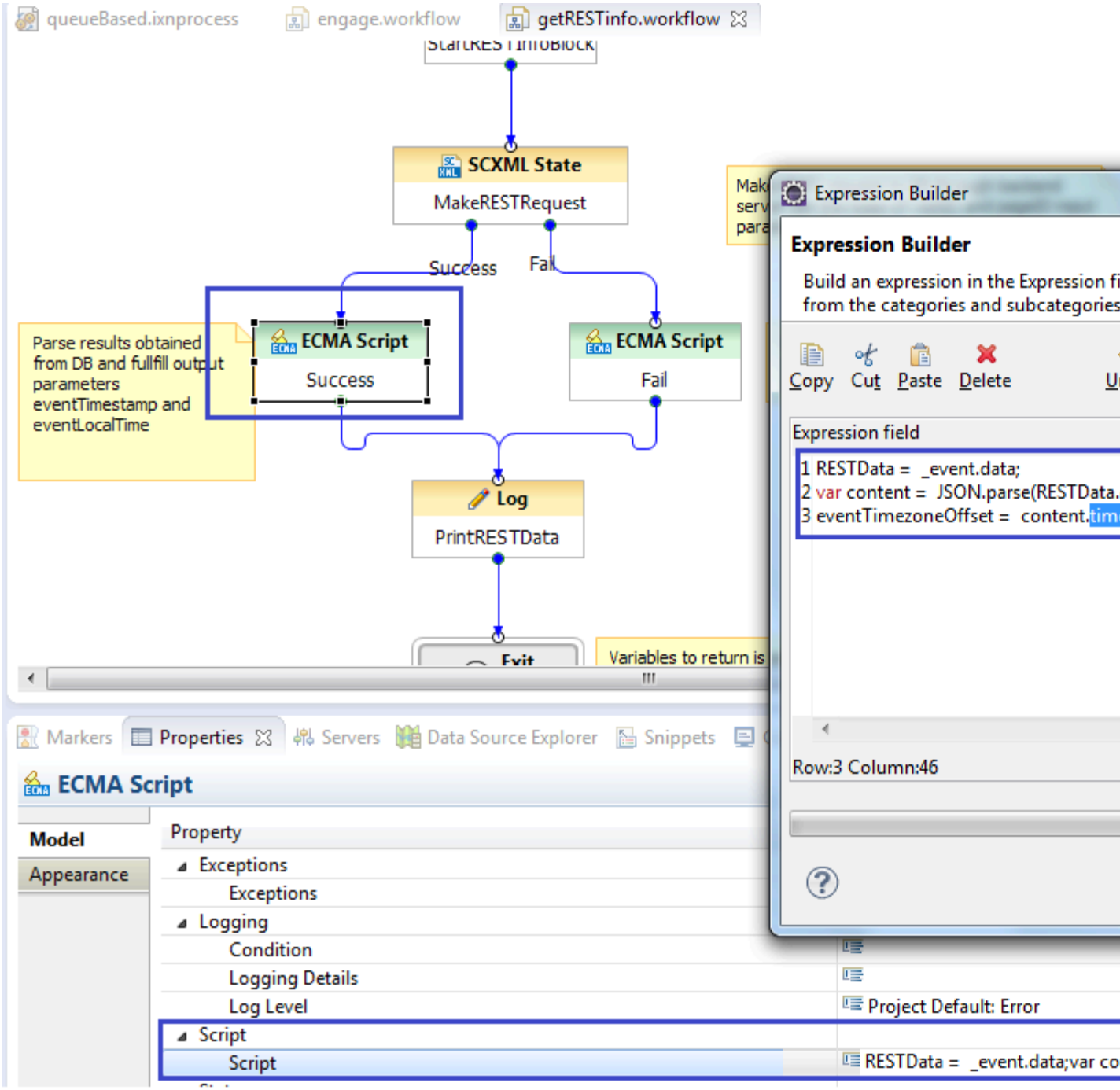
1. Use **SCXML State** block in order to make the REST request with specified parameters.



Use the State block to make REST requests

Note: BackendURL and **visitID** parameters are passed from the parent workflow into this sub-flow.

2. Parse response to the REST request. After the response is successfully obtained, it should be parsed in order to extract required data. In this example, the **timezoneOffset** parameter is obtained from the data of the VisitStarted event:



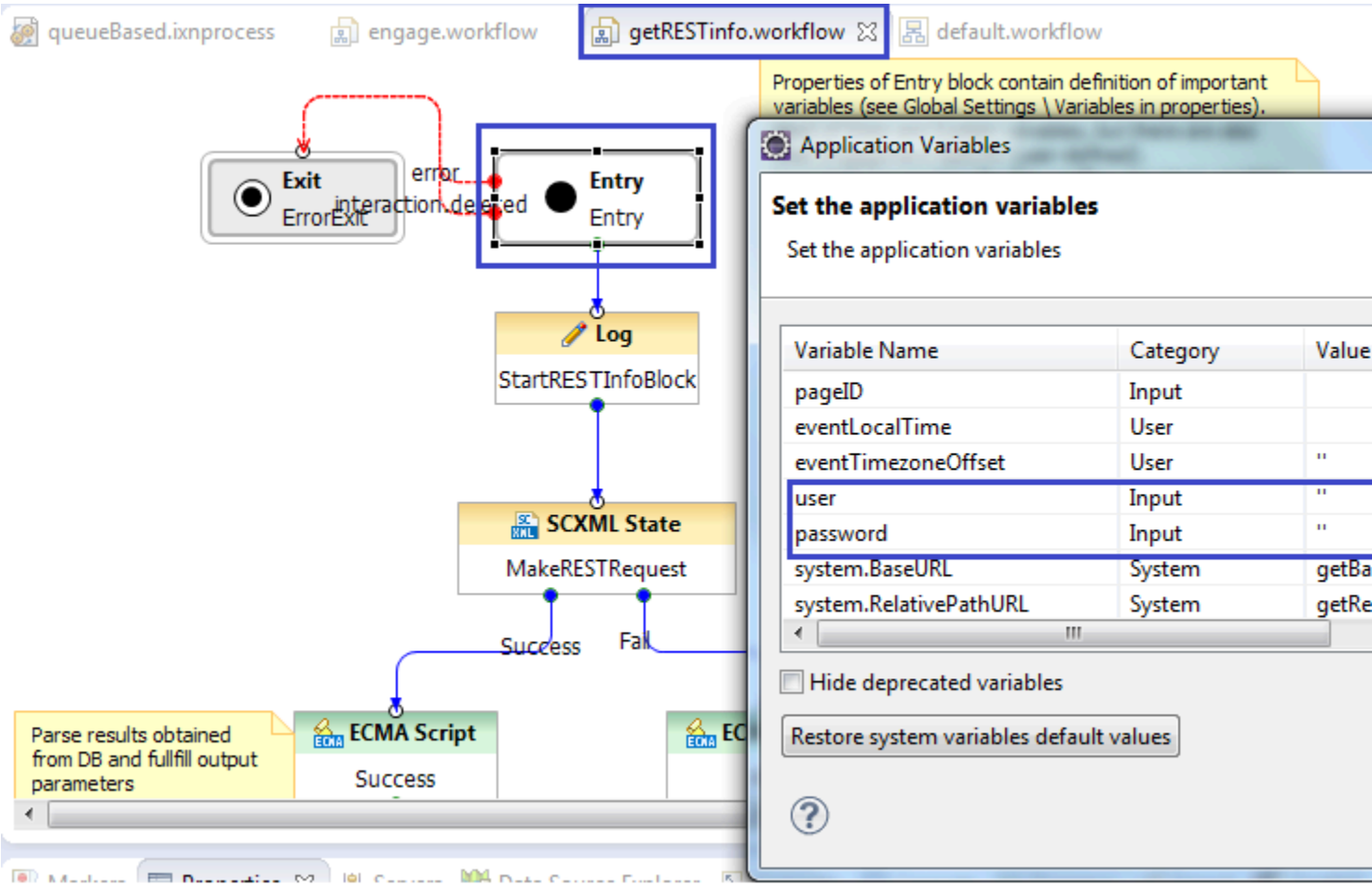
Parse the response to the REST request

Note: Alternatively, instead of the **SCXML State** block, you can use a **Web Request** or **Web Service** block. In this case, Composer requires this logic to be hosted as a web application, which means the entire Composer project must be hosted outside of the Web Engagement application. With Composer, you can export the project as a web application in WAR format. This approach is not used in out-of-the-box strategies.

Configure Authentication in the out-of-the-box SCXML Strategy

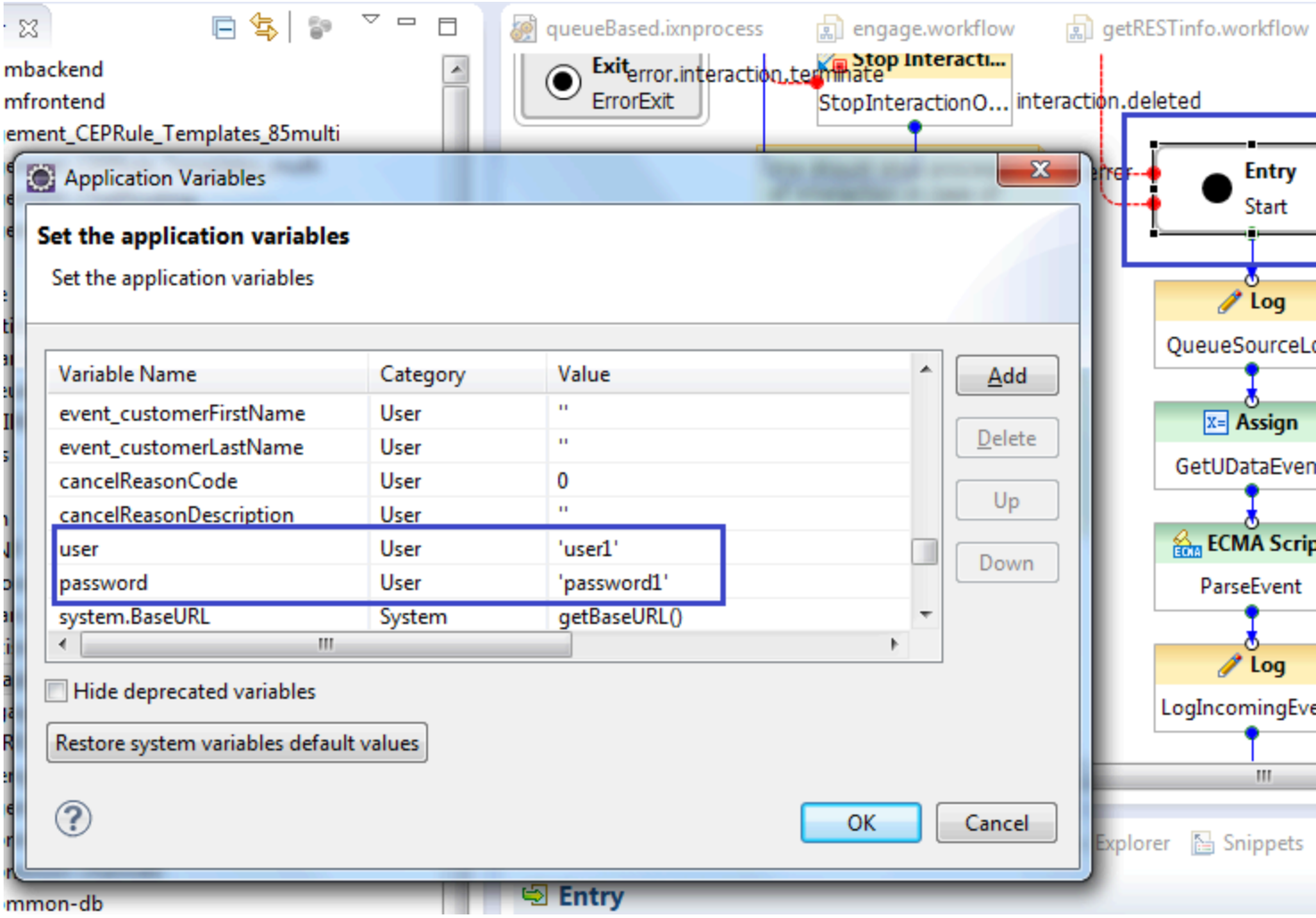
Genesys Web Engagement 8.1.2 provides basic access authentication on the base of providing username/password pairs.

Username and password parameters, used in the **SCXML State** block, are passed into **getRESTInfo** workflow from the parent workflow:



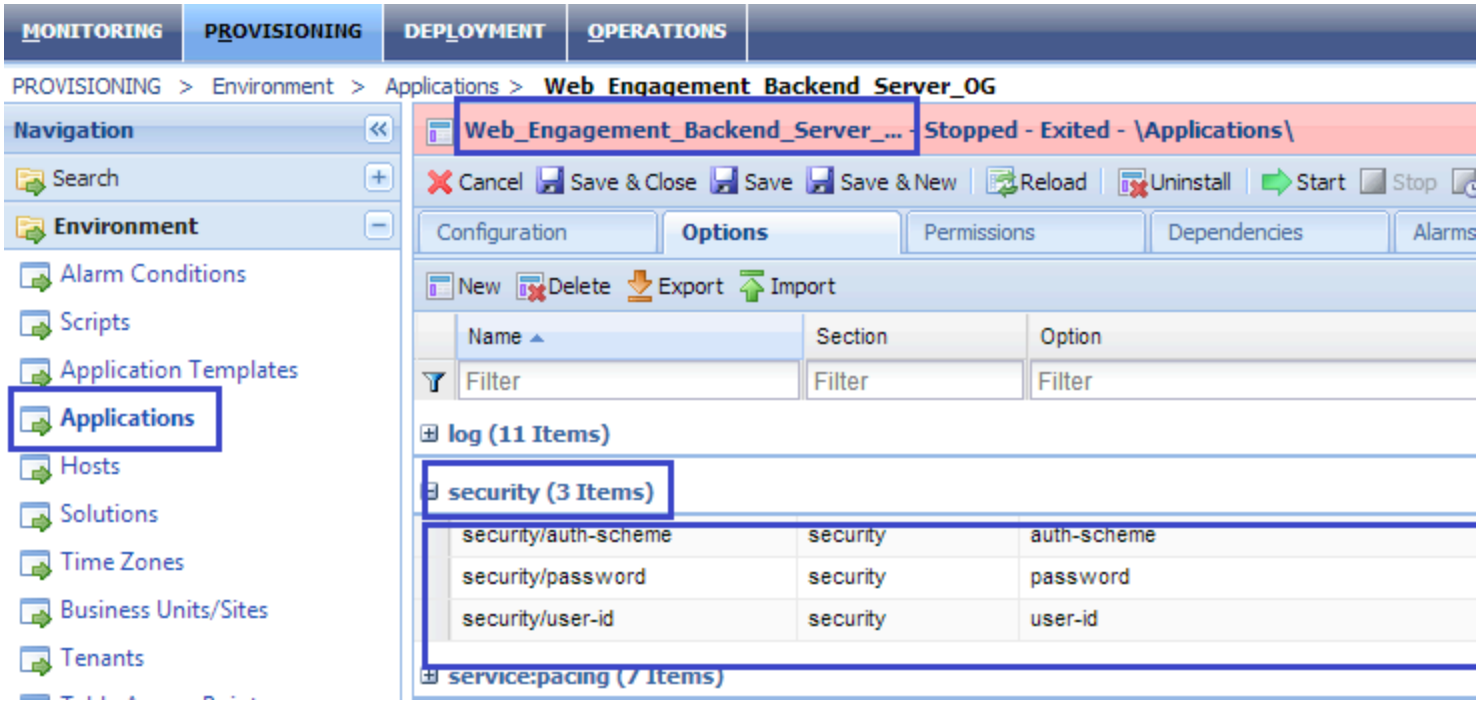
The username and password application variables in **getRESTInfo.workflow**.

The username and password parameters are specified in variables of the **Entry** block in **default.workflow**:



The username and password application variables in the **default.workflow**.

You must check that these credentials are compliant with credentials specified in the security section of the Backend Server options:



The username and password are specified in the security section

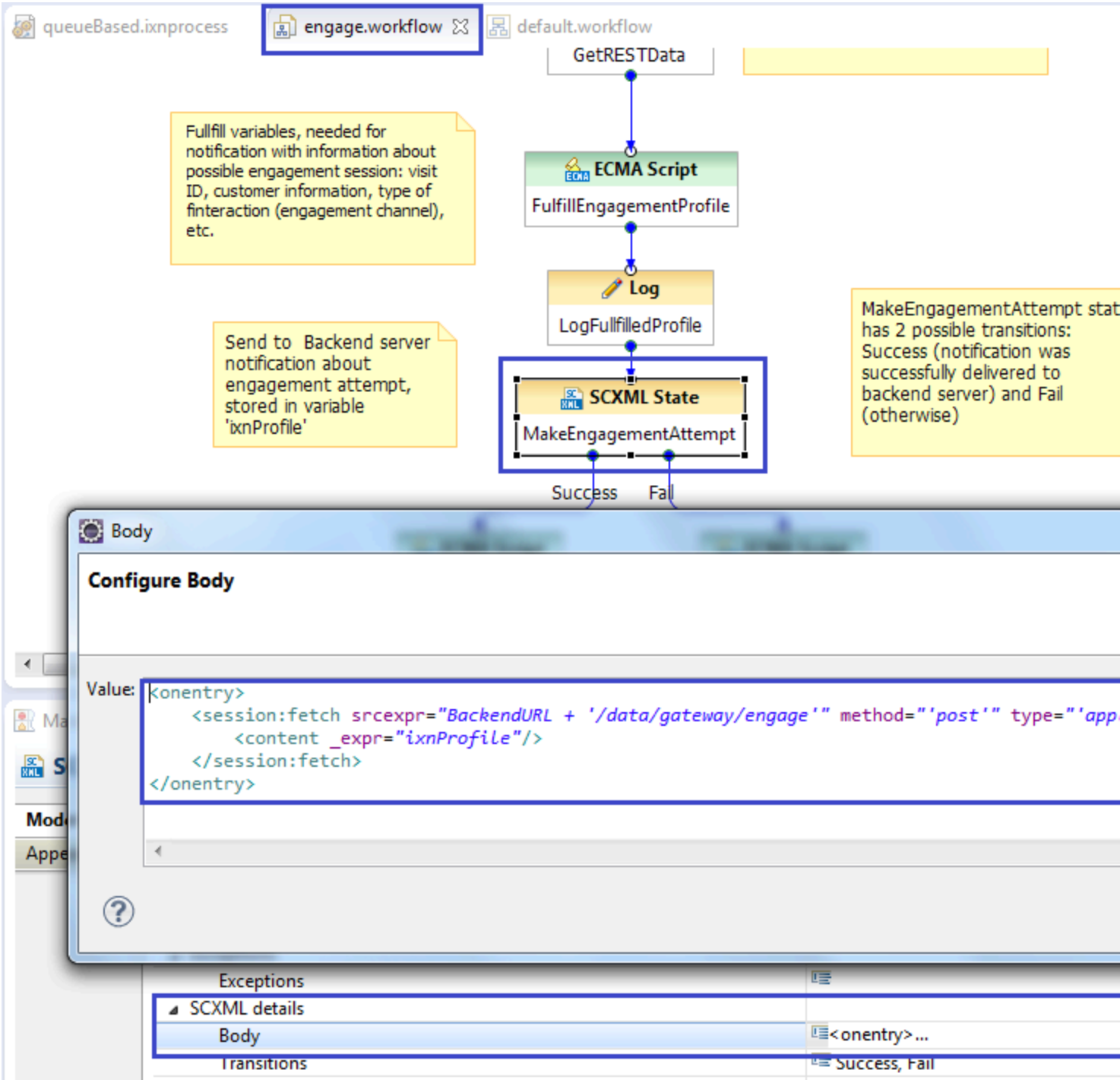
See Configuring Authentication for details.

Start Engagement as a Result of the Engagement Logic Strategy

Sending the "start engage" Request to the Backend Server

The special workflow **engage.workflow** notifies the Backend Server about the `start engage` command.

Notification of the Backend Server is executed through the REST request using the **SCXML State** block:



The REST request notifies the Backend Server

Note: Authentication aspects shown here are the same in **getRESTInfo.workflow.**

Fulfilling IxnProfile for "start engage" Request

Take note of the **IxnProfile** structure, which is passed in REST request to the Backend Server. This structure is fulfilled in the **ECMA Script** block called FulfillEngagementProfile.

The following object is sent to the Browser:

```
ixnProfile = {
'data': data
}
```

Consider the structure of the data object:

```
var data = {
    'chat': engageProfile,
    'event': event,
    'notification': notification_message
}
```

As you can see, there are three fields:

- chat — represented by the variable **engagementProfile**.
  - Content of this variable will be considered below. You can change the content of this variable if the SCXML strategy worked in the area of visitor identification.
  - It is not recommended to change it if related items are not a part of your modified strategy.
- event — this is a technical field, which provides for the Backend Server possibility to identify an event, on the base of which the engagement decision was done.
  - You should not change the structure or content of this member.
- notification — represented by the variable **notification_message**.
- Structure of the notification message is described in Chat Invitation Message and Callback Invitation Message.

Consider the structure of **engagementProfile** variable:

```
var engageProfile = {
    'visitId': event_visitID,
    'nickName': profile.FirstName,
    'firstName': profile.FirstName,
    'lastName': profile.LastName,
    'userState': state,
    'userId': event_customerID,
    'ixnId': system.InteractionID
};
```

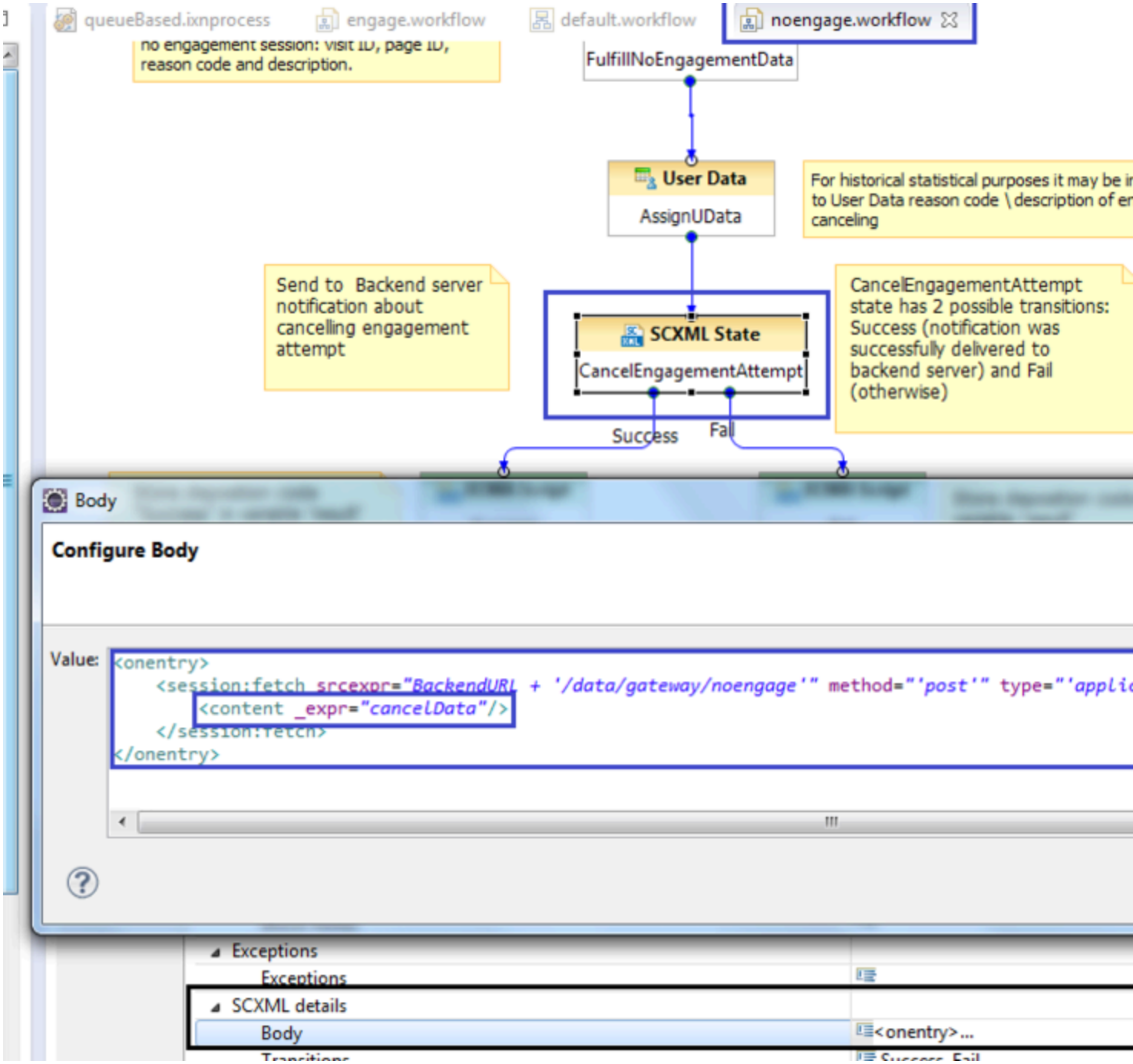
You can change the fields **nickName, firstName, lastName** and **state** in the case of additional work being executed in the visitor identification area. In this case, the Backend Server applies passed values to the identity record of specified **visitId**.

The following states are allowed: Authenticated, Recognized, and Anonymous.

Cancelling Engagement as a Result of the Engagement Logic Strategy

Sending "cancel engagement" to Backend Server

This is similar to sending **start engage**, request **cancel engagement**; it also uses the **SCXML State** block to trigger a REST request to the Backend Server:

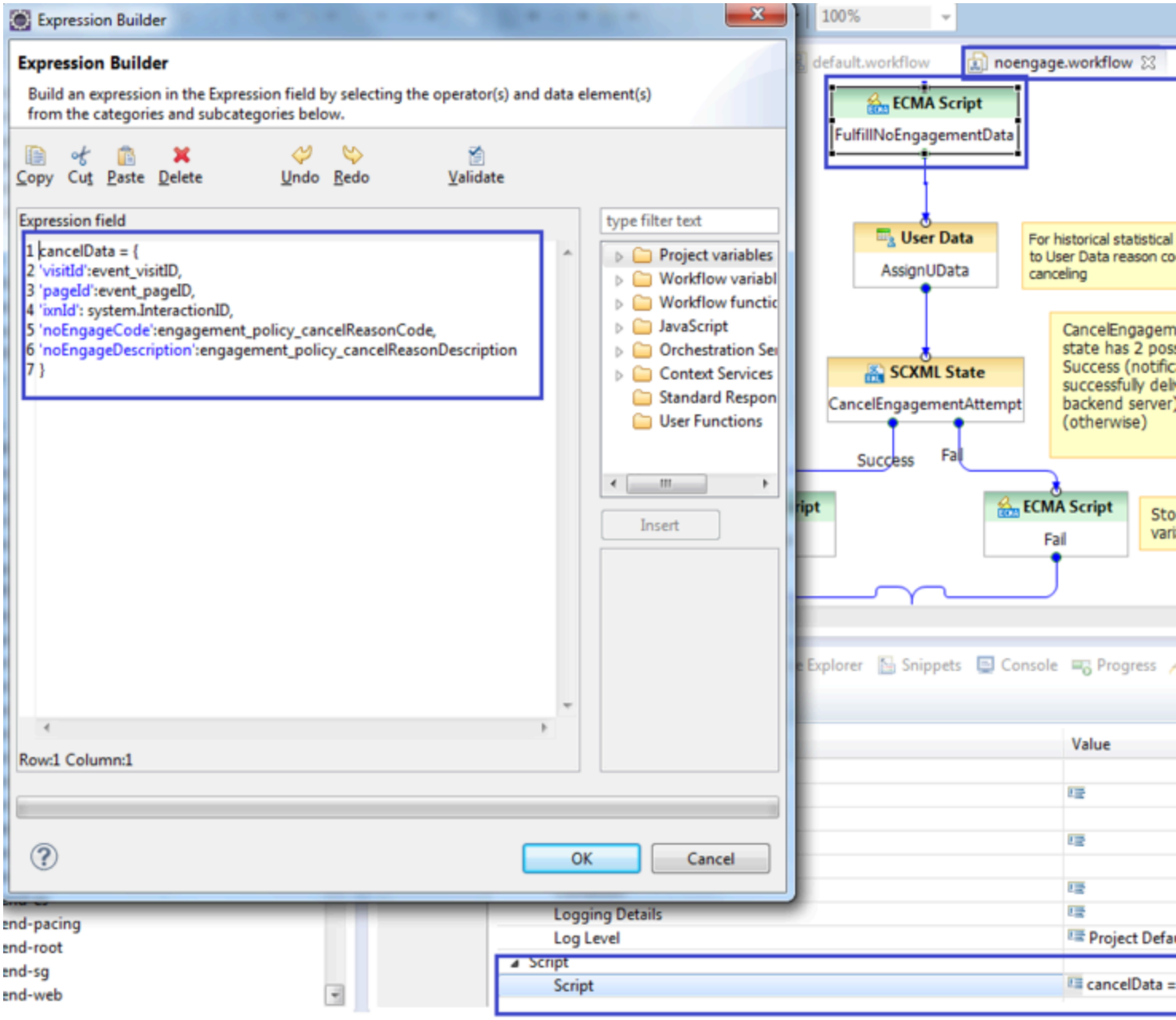


The REST request cancels the engagement

Security (authentication) aspects are the same as described in the **getRESTInfo.worfkflow**'.

Fulfilling "no engage" Data

no engage data contains five mandatory fields:



The "no engage" fields

Cleaning Interaction Process

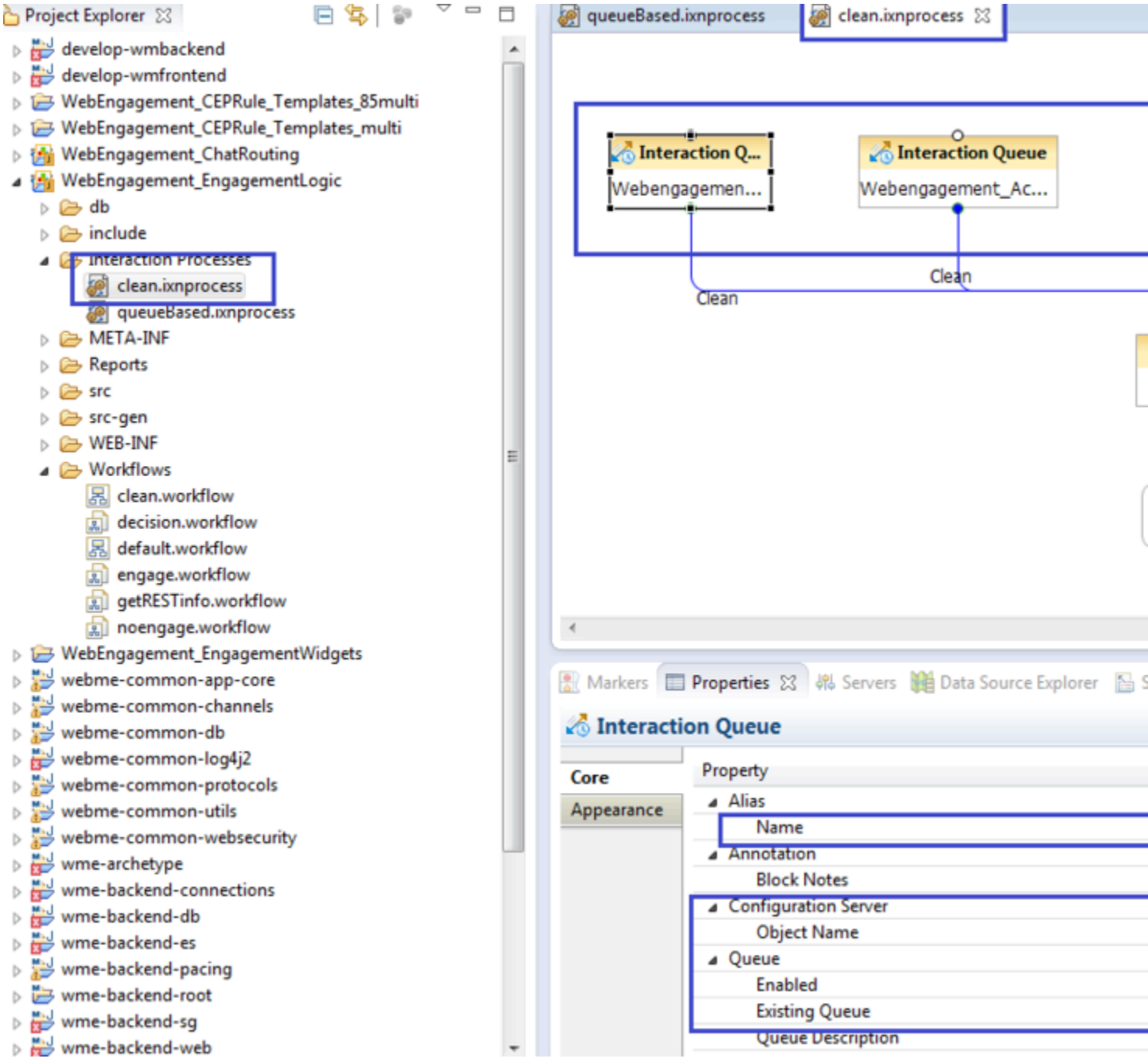
The interaction, for various reasons, might be stuck in one of the Interaction Queues and fall out of processing. For example:

- Visitor obtained engagement invitation. This means that the webengagement interaction was put into

Webengagement_Accepted queue.

- Power-off appeared on visitor's host, so the answer (Accept, Reject, or Timeout) was not delivered to Genesys Web Engagement.

In this case, you need to define the cleaning process, which is also built on the top of ORS strategies.



The **clean.ixnprocess**

As shown, the same cleaning process is applied for all Web Engagement-related Interaction Queues. The only exemption is Webengagement_Qualified queue; this queue is not cleaned by the strategy.

The cleaning task in this queue is executed in scope of major Engagement Logic interaction process.

Cleaning Interaction Workflow

Out-of-the-box cleaning workflow is short and straightforward. It contains one block only: **stop interaction**.

Propagating Data from Engagement Logic strategy into Chat Routing Strategy

Use Case Description

In the routing process, it often makes sense to use business data from events that are produced on the browser side. This data is propagated by the Backend Server to the webengagement interaction automatically, but you can also propagate it to the chat or web callback interactions.

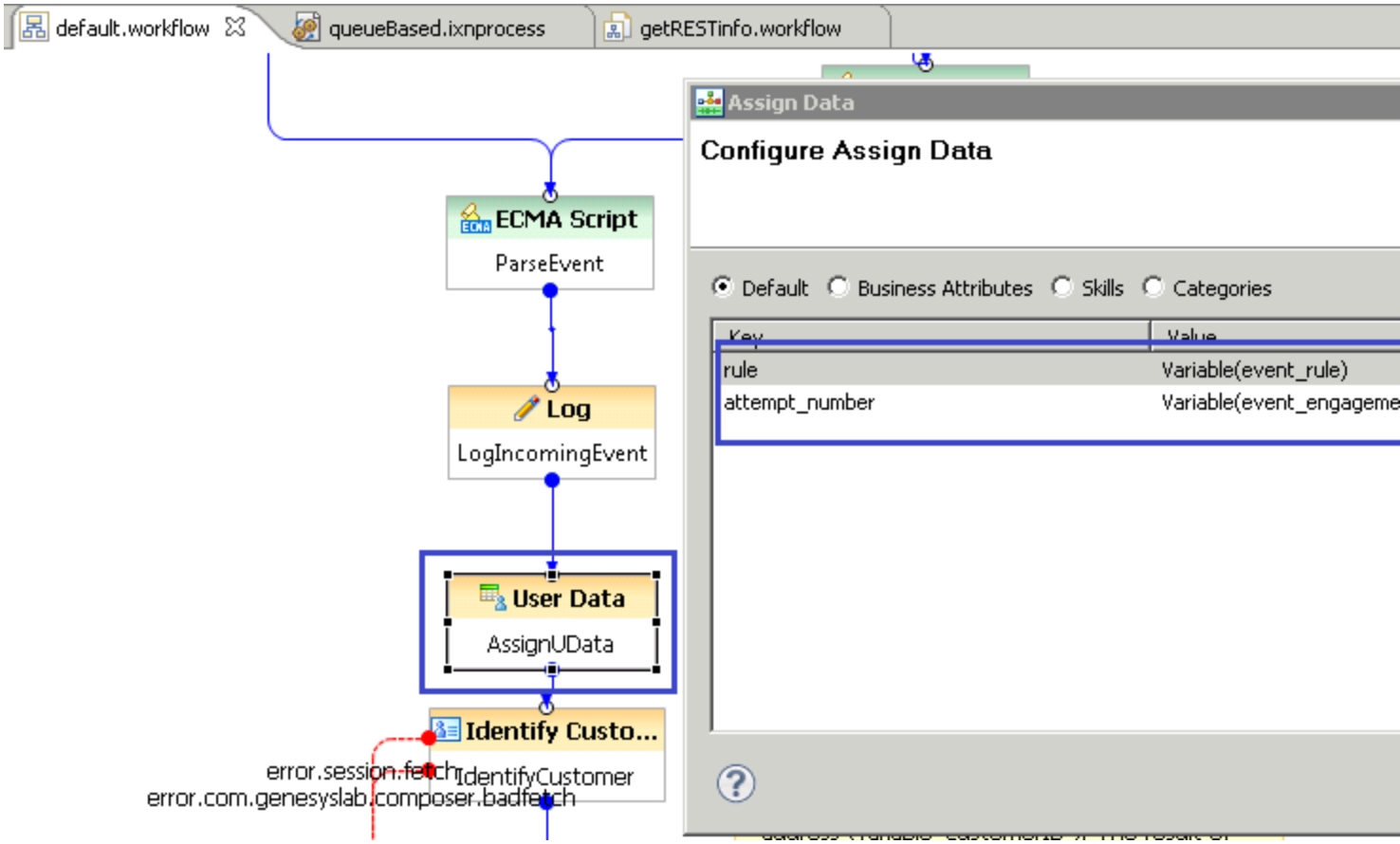
For example:

- Business data produced on the page provides information about language.
- This information is passed as a sub-key of the **jsonEvent** key into the webengagement interaction.
- During the Engagement Logic strategy, language information is re-attached and propagated to the chat interaction.
- The Chat Routing strategy reads language information from the chat interaction and decides into which group to route the chat interaction.

The following are details of the described data propagation.

Attach UserData to the webengagement Interaction

All data that comes from System events is stored in the Open Media webengagement interaction as a KVlist under the key **jsonEvent**. You can access this data from the engagement strategy. If you want to store this user data and then copy it into the chat or webcallback engagement interaction, you must attach it manually to the Open Media webengagement interaction in the engagement strategy. For example, you can do this with the **User Data** block:



Assigning User Data

> ## Important
>
> Genesys recommends that you collect all the data you need and attach it to the interaction in a single **Assign Data** block. You should avoid using multiple **Assign Data** blocks unless is it absolutely necessary.

Control Copying UserData from webengagement Interaction to the Chat (or web callback) Interaction

When a chat or web callback interaction is created, GWE attaches the UserData available in its parent Open Media **webengagement** interaction. You can control how this data is attached by using the wes.connector.interaction.copyUserData option in the **[service:wes]** section of the Backend Server application. This option has three modes:

- Copy all UserData
- Do not copy UserData
- Copy only specific KV pairs from UserData

The following tables provide example values for the wes.connector.interaction.copyUserData option. In these examples, the Open Media webengagement interaction UserData contains the keys **ORS Data, rule, strategy, some data**.

Value of wes.connector.interaction.copyUserData	Data in the engagement interaction
all	All keys are copied: ORS Data, rule, strategy, some data.
no	No keys are copied.
rule;strategy	The rule, strategy keys are copied.
blank or empty	If the value of wes.connector.interaction.copyUserData is absent or has an empty value, no keys are copied.
my_key1;ORS Data	The ORS Data key is copied. my_key1 is ignored because it is not part of the keys in the Open Media webengagement interaction UserData.

Accessing Pacing Information from the Engagement Logic Strategy

In release 8.1.2, Web Engagement provides the Engagement Logic strategy with pacing data for the chat and web callback channels. You can access pacing information in two ways:

- Through the consolidated channel capacity (measured in the number of "allowed" interactions).
- Through detailed information for each channel, which contains capacity (measured in the number of "allowed" interactions) for each particular group in a channel.

> **Important**
>
> The pacing information available to the Engagement Logic strategy is different from the information returned from the Pacing API. You should evaluate each type of pacing information carefully before deciding how to use it.

Pacing information is added to **webengagement** open media interaction User Data by the Backend Server. This information can then be read in the SCXML strategy — see Main Interaction Process and Workflow for an example. The information is located (among other specific data, like data provided in business-event) in the User Data of the webengagement interaction, under the **jsonEvent** key. This key contains the JSON object, which should be parsed prior to access information.

In the SCXML strategy, you can access **jsonEvent** data in the following way:

```
var jsonEvent = _genesys.ixn.interactions[system.InteractionID].udata.jsonEvent;
var eventData = JSON.parse(jsonEvent);
```

Understanding How the Pacing Algorithm Works

A dedicated pacing algorithm serves each particular group of agents, so if you have 2 chat-oriented and 1 web callback-oriented group of agents, there will be 3 instance of the pacing algorithm (1 for each group).

The agent availability on the specific channel is calculated taking into account the following:

- The agent state on the particular media (chat and web callback are different)
- Capacity rules.

For example, consider an agent who has a capacity rule for 2 chat interactions. In this scenario, the following statements are true:

- Agent is Ready and has no interactions in progress. In this case, the agent is treated as 2 Ready agents with a capacity rule of 1.
- Agent is Ready and has one interaction in progress. In this case, the agent is treated as 1 Ready agent with a capacity of 1.
- Agent is Ready and has two interactions in progress. In this case, the agent is treated as 0 Ready agents with a capacity of 1.
- Agent is Not Ready (count of interactions in progress does not matter). In this case, agent is treated as 0 Ready agents with a capacity of 1.

The agent availability on the specific channel is also handled differently in the two main pacing algorithm methods, SUPER_PROGRESSIVE and PREDICTIVE_B.

The SUPER_PROGRESSIVE method consumes the following major parameters:

- The number of Ready agents in the group.
- The number of pending (waiting for answer) interactions.
- HitRate - the percentage of accepted invitations compared to the general number of proposed engagement invitations.

> **Important**
>
> It is important to remember that the values of these parameters are continuously changing.

Consider the following example: There are 7 Ready agents (each with a capacity rule of 1), the number of pending interactions is 5, and the HitRate is 0.05.

In this case, the pacing algorithm might predict the number of allowed interactions approximately as (7 / 0.05 - 5) = 135.

> **Important**
> This example is intended to provide a basic idea of how the pacing algorithm works. The finer details are more complex.

The PREDICTIVE_B method consumes the following major parameters:

- The number of logged in agents in the group.
- The Average handling time of interactions. For example, the average duration of a chat session with visitors.
- HitRate - the percentage of accepted invitations compared to the general number of proposed engagement invitations.

> **Important**
> It is important to remember that the values of these parameters are continuously changing.

This algorithm is more complex than SUPER_PROGRESSIVE, but the general information described for SUPER_PROGRESSIVE also applies to PREDICTIVE_B: The number of 'allowed' interactions will significantly exceed the number of Logged In agents (depending, first of all, on the HitRate parameter).

Consolidated Pacing Information by Channel

Capacity for the chat channel is available in the **data.chatChannelCapacity' *field (called data.chatLoad* before version 8.1.200.26), and capacity for the web callback channel is available in the** *data.webcallbackChannelCapacity* field (called **data.voiceLoad** before version 8.1.200.26).

For example:

```
var jsonEvent = _genesys.ixn.interactions[system.InteractionID].udata.jsonEvent;
var eventData = JSON.parse(jsonEvent);

var chatChannelCapacity = eventData.data.chatChannelCapacity;
var webcallbackChannelCapacity = eventData.data.webcallbackChannelCapacity;
```

Detailed Pacing Information

Detailed pacing information is available as a nested JSON object with the following structure:

```
pacing: {
  channels :
  [
    {
       name: <name of this channel>,
       groups:
```

```
      [
         {
           name: <name of this group>,
           capacity: <count of allowed interactions for this group>,
           reactiveTrafficRatio: <portion of inbound chat\webcallback traffic that should be
'left' in the system>
         },
         ...
      ],
      capacity: <count of allowed interactions for this channel>
   },
   ...
  ]
}
```

The names of the pacing information fields were changed in release 8.1.200.26. See the table below for details.

Name prior to 8.1.200.26	Name as of 8.1.200.26	Description
channelName / groupName	name	Name of the channel (or group, depending on the type of container object).
intrNumber	capacity	The number of 'allowed' interactions for a channel (or group, depending on the type of container object).
reactiveTrafficRatio	reactiveTrafficRatio	Portion of inbound chat traffic that should be 'left' in the system. Valid values: from 0 to 1 For example, reactiveTrafficRatio 0.8 means that only 8 of 10 'reactive' chat interactions should be 'allowed' .

You can access detailed information in the Engagement Strategy SCXML as follows:

```
var jsonEvent = _genesys.ixn.interactions[system.InteractionID].udata.jsonEvent;
var eventData = JSON.parse(jsonEvent);

var detailedPacing=JSON.parse(eventData.data.pacing);
var event_chatEnglishCapacity = pacing.channels[0].groups[0].capacity;
var event_chatDutchLoadCapacity = pacing.channels[0].groups[1].capacity;
```

In the example above, IDs (0, 1, and so on) in the arrays are used for sample purposes only. You should use the specific names of the channels and groups to extract the data you need.

Example of Using Pacing Information

Agents

Consider the following scenario where there are four chat and voice groups with agents in each group:

- English Language Chat Group = Adam (logged in and ready) and Anna (logged in, not ready)

- Dutch Language Chat Group = Bart (NOT logged in) and Berta (NOT logged in)
- English Language Voice Group = Adam (logged in and ready) and Amanda (logged in and ready)
- Dutch Language Voice Group = Dan (logged in, ready)

The following group configuration options are set on the Backend Server application:

- pacing.connector.chatGroup = English Chat Group;Dutch Chat Group
- pacing.connector.voiceGroup = English Voice Group;Dutch Voice Group

Customers

On the customer-facing website, two events are triggered simultaneously:

- **Chris** triggers a Hot Lead event on an English page.
- **Merijn** triggers a Hot Lead event on a Dutch page.

Pacing information

When events are triggered simultaneously, pacing information is the same. In this scenario, the SUPER_PROGRESSIVE algorithm is used and the following parameters were true at the moment the events were triggered:

- English Chat Ready agents: 1
- Dutch Chat Ready agents: 0
- English Voice Ready agents: 2
- Dutch Voice Ready agents: 1
- HitRate: 0.2
- Pending engagement invites: 0
- Reactive traffic is turned off

In this case, the results might look like this:

```
...
chatChannelCapacity : 5,
webcallbackChannelCapacity : 16,
pacing: {
  channels :
  [
    {
       name: "chat",
       groups:
       [
          {
            name: "English Language Chat Group",
            capacity: 5,
            reactiveTrafficRatio: 0
          },
          {
            name: "Dutch Language Chat Group",
```

```
            capacity: 0,
            reactiveTrafficRatio: 0,
         }
      ],
      capacity: 5
   },
   {
      name: "webcallback",
      groups:
      [
         {
           name: "English Language Voice Group",
           reactiveTrafficRatio: 0,
           capacity: 11
         },
         {
           name: "Dutch Language Voice Group",
           reactiveTrafficRatio: 0,
           capacity: 5
         }
      ],
      capacity: 16
   }
  ]
}
```

Possible Engagement Logic SCXML flows

In this scenario, the following SCXML flows are possible for the two customers, Chris and Merijn:

- **Chris**
  We can extract the capacity for the "English Language Chat Group" (5) and "English Language Voice Group" (11) from the pacing data.

  In the decision workflow, it is possible to engage Chris on the chat or web callback channel. It is also possible to show him a modified invitation, where he can explicitly choose chat or web callback.

- **Merijn**
  We can extract the capacity for the "Dutch Language Chat Group" (0) and "Dutch Language Voice Group" (5) from the pacing data.

  In the decision workflow, it is possible to engage Merijn on the web callback channel only.