Developer's Guide

Customizing the Engagement Strategy

5/4/2025

Customizing the Engagement Strategy

Contents

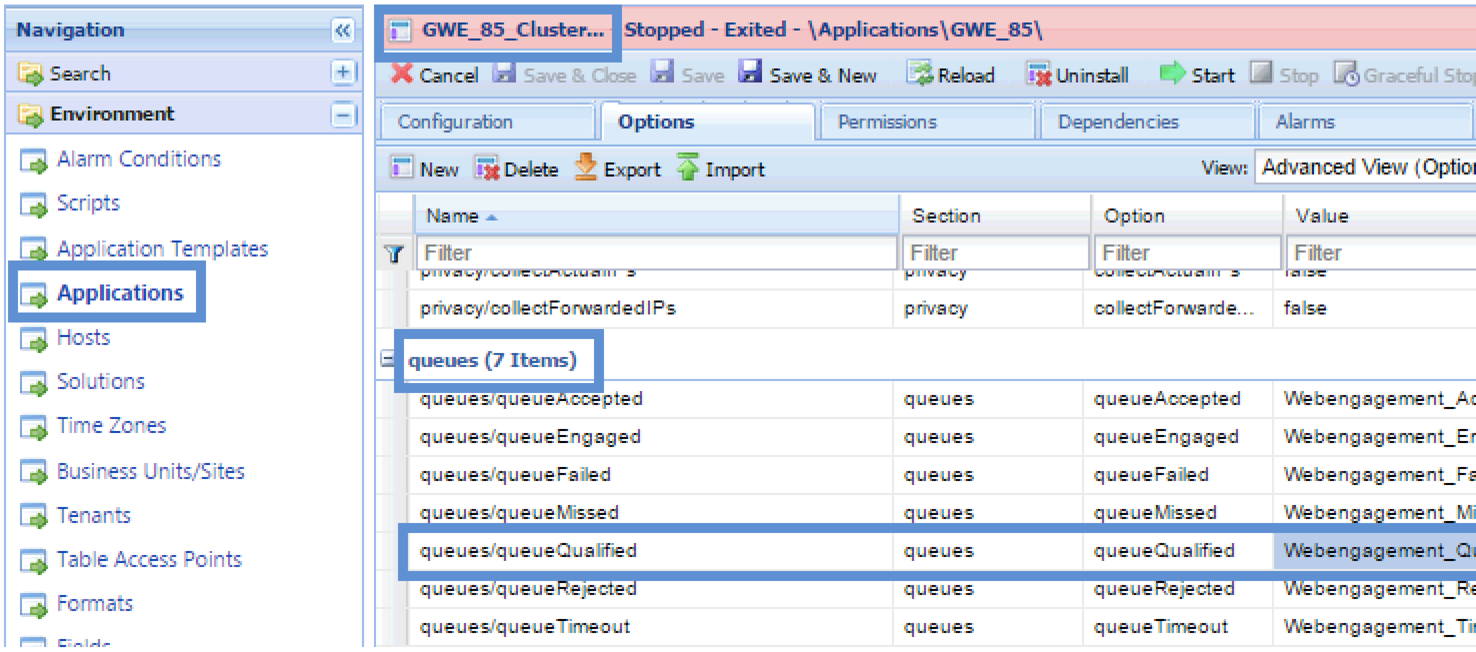
When you create your Web Engagement application, Genesys Web Engagement also creates default Engagement Logic and Chat Routing SCXML strategies in the **\apps*application_name*\resources_composer-projects** folder. Orchestration Server (ORS) uses these strategies to decide whether and when to make a proactive offer and which channels to offer (chat or other custom widget, for example - an advertisement).

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 Web Engagement 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*\resources_composer-projects\WebEngagement_EngagementLogic\**. Refer to the sections below for details about the Engagement Logic strategy and how it can be modified.

Main Interaction Process and Workflow

When Genesys Web Engagement creates an engagement attempt, the Web Engagement Server creates an Open Media interaction of type **webengagement** and places it into the interaction queue specified by the 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.



Name	Section	Option	Value
privacy/collectForwardedIPs	privacy	collectForwarde...	false
queues (7 Items)			
queues/queueAccepted	queues	queueAccepted	Webengagement_Ac
queues/queueEngaged	queues	queueEngaged	Webengagement_Er
queues/queueFailed	queues	queueFailed	Webengagement_Fa
queues/queueMissed	queues	queueMissed	Webengagement_Mi
queues/queueQualified	queues	queueQualified	Webengagement_Qu
queues/queueRejected	queues	queueRejected	Webengagement_Re
queues/queueTimeout	queues	queueTimeout	Webengagement_Ti

The Interaction Queue

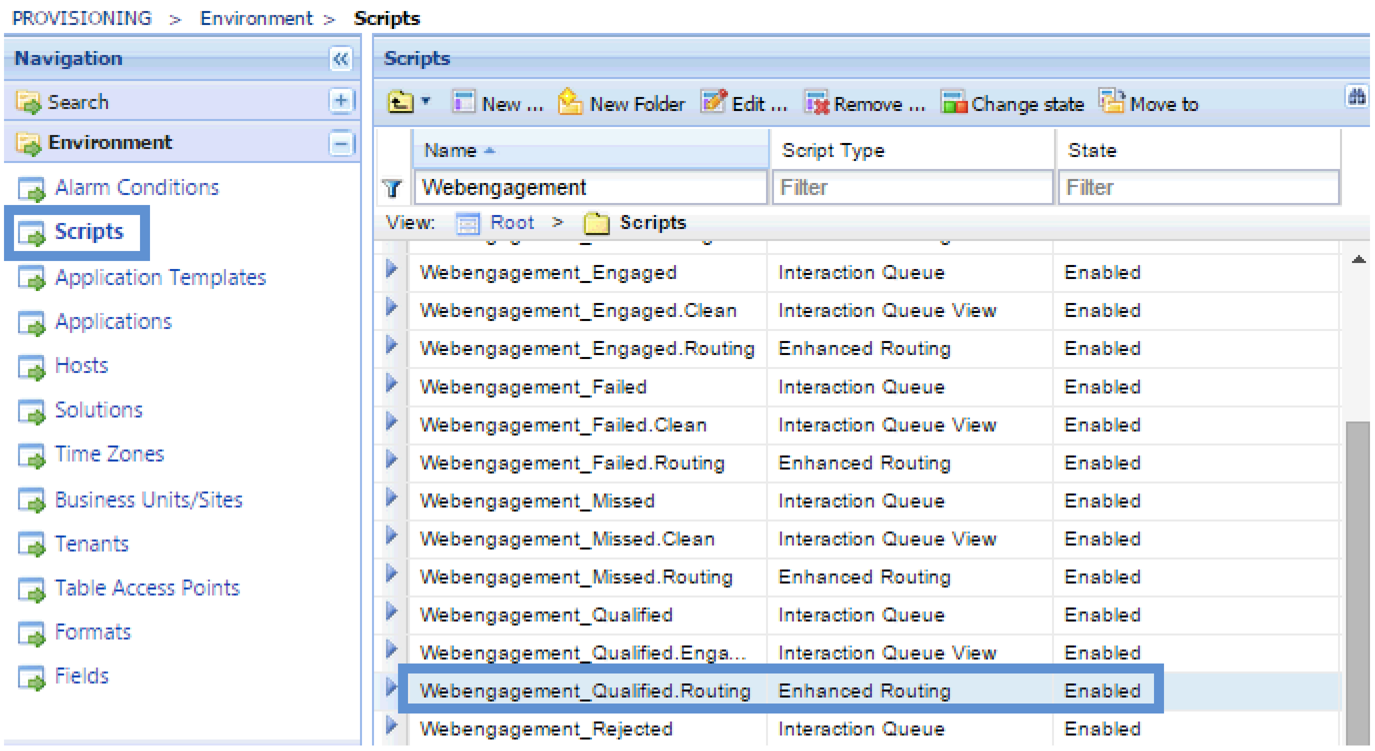
Passing Parameters into the Engagement Logic Strategy

When Genesys Web Engagement creates an engagement attempt, the Web Engagement Server creates an Open Media interaction of type **webengagement** and places it into the Interaction Queue specified by the 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 Web Engagement 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 Web Engagement 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 Web Engagement 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 Web Engagement Server.

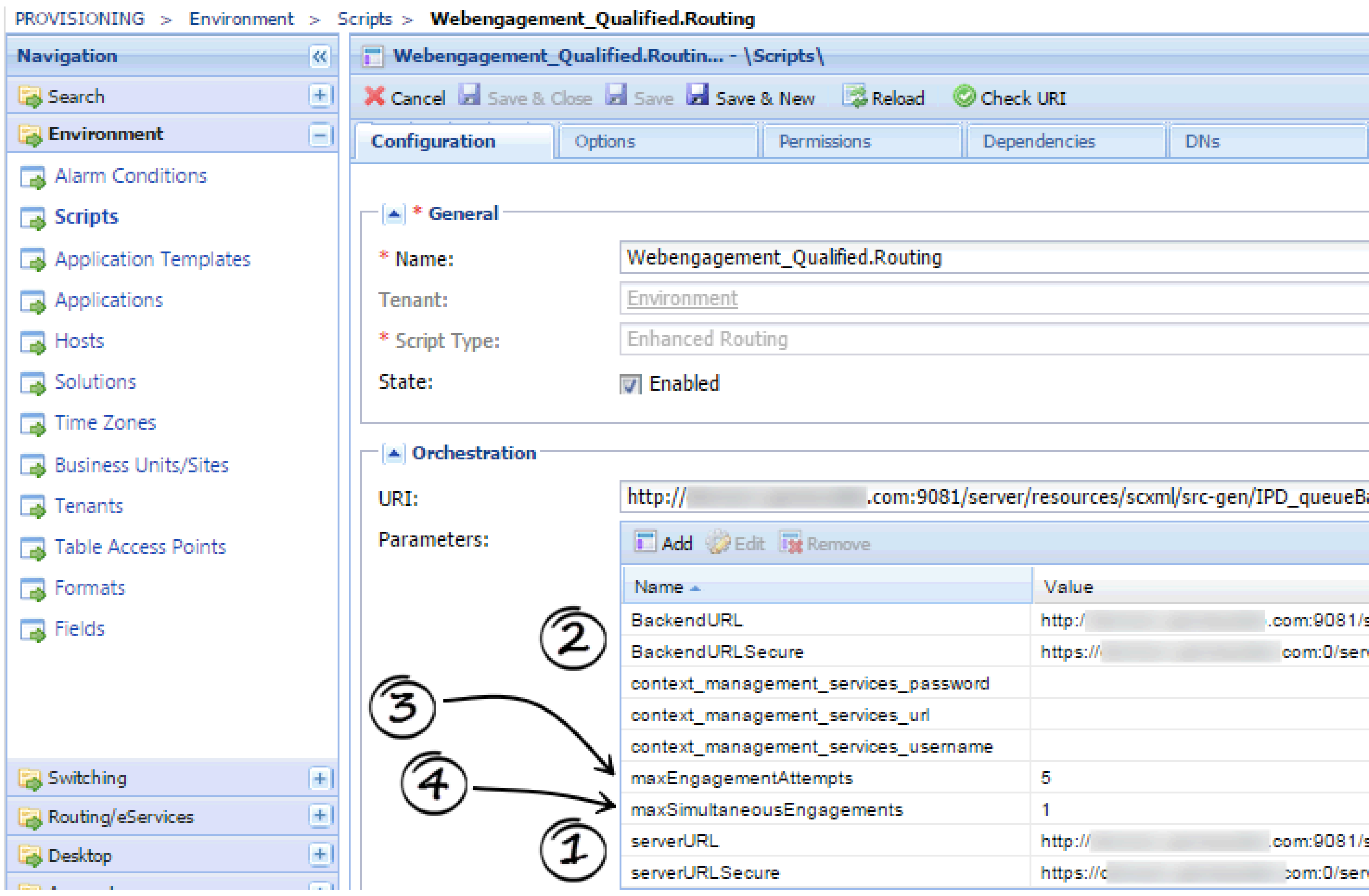
The parameters are passed to ORS through the Enhanced Routing script object Webengagement_Qualified.Routing that is associated with the Webengagement_Qualified Interaction Queue.



Name	Script Type	State
Webengagement	Filter	Filter
Webengagement_Engaged	Interaction Queue	Enabled
Webengagement_Engaged.Clean	Interaction Queue View	Enabled
Webengagement_Engaged.Routing	Enhanced Routing	Enabled
Webengagement_Failed	Interaction Queue	Enabled
Webengagement_Failed.Clean	Interaction Queue View	Enabled
Webengagement_Failed.Routing	Enhanced Routing	Enabled
Webengagement_Missed	Interaction Queue	Enabled
Webengagement_Missed.Clean	Interaction Queue View	Enabled
Webengagement_Missed.Routing	Enhanced Routing	Enabled
Webengagement_Qualified	Interaction Queue	Enabled
Webengagement_Qualified.Enga...	Interaction Queue View	Enabled
Webengagement_Qualified.Routing	Enhanced Routing	Enabled
Webengagement_Rejected	Interaction Queue	Enabled

The Webengagement_Qualified.Routing Script Object

There are several parameters specified by default, as shown in the following image.



* Name:	Webengagement_Qualified.Routing
Tenant:	Environment
* Script Type:	Enhanced Routing
State:	✓ Enabled
URI:	http://[redacted].com:9081/server/resources/scxml/src-gen/IPD_queueB

Parameters:

Name	Value
BackendURL	http:/[redacted].com:9081/s
BackendURLSecure	https://[redacted]com:0/ser
context_management_services_password	
context_management_services_url	
context_management_services_username	
maxEngagementAttempts	5
maxSimultaneousEngagements	1
serverURL	http://[redacted].com:9081/s
serverURLSecure	https://[redacted]com:0/ser

The Webengagement_Qualified.Routing Parameters

The first set of parameters, **(1) serverURL** and **serverURLSecure** correspond to the **(2) BackendURL** and **BackendURLSecure** parameters used in 8.1.2, and are not available anymore. You can also set **(3)** the maximum number of engagement attempts and **(4)** the maximum number of simultaneous engagements.

In cases where you need a separate address for chat processing, use the **mediaServerURL** parameter. This parameter is similar to the **serverURL** parameter but is used to specify a separate URL to be used only for chat processing. This can be useful in situations where:

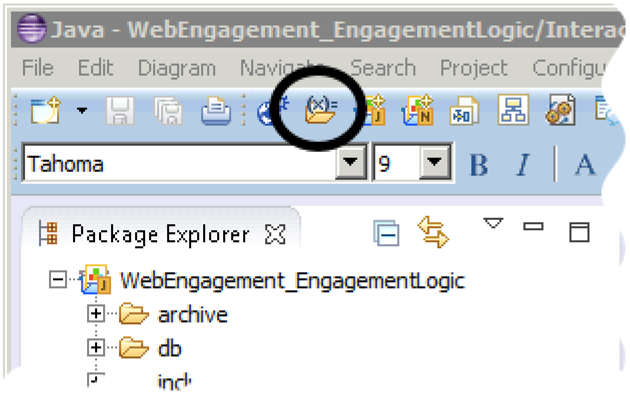
- Event traffic uses a non-secure server (as specified by the **serverURL** parameter), but you need a secure connection for your chat traffic (in which case **mediaServerURL** will specify an HTTPS endpoint)
- Event traffic is processed on one port, but chat traffic needs to be processed on a second port on the same host

The Engagement Logic strategy has two interaction processes:

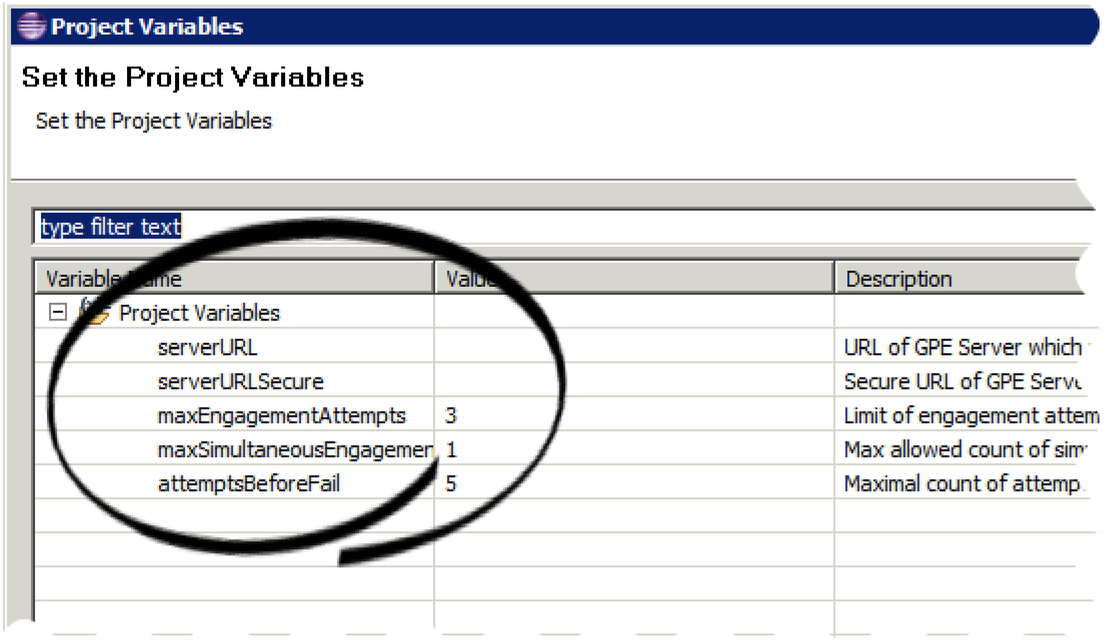
- **clean.ixnprocess** — This process is explained in Cleaning Interaction Process
- **queueBased.ixnprocess** — This process features the major logic for the strategy.

In this section, we will consider the second one.

To access the above-mentioned parameters from within Composer, use the Composer **Access Project Variables** button shown in the following image. **Note:** In order to access Project Variables, your current tab in Composer must display Interaction Process (not Workflow).



This button opens a window containing the variables we are currently interested in:


Project Variables

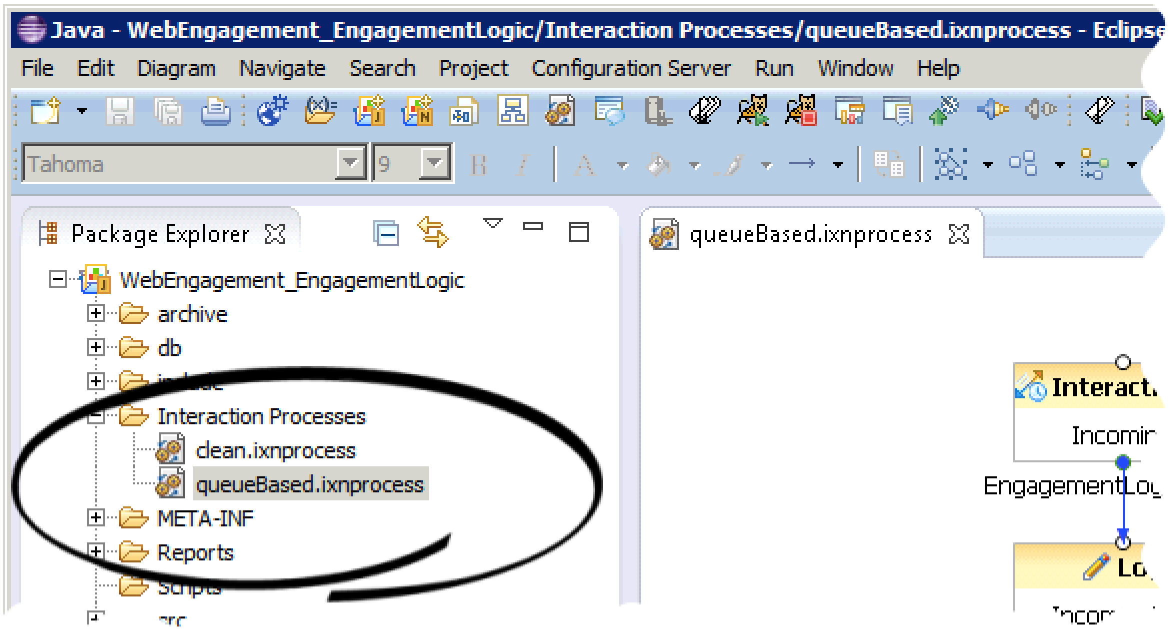
Set the Project Variables

Set the Project Variables

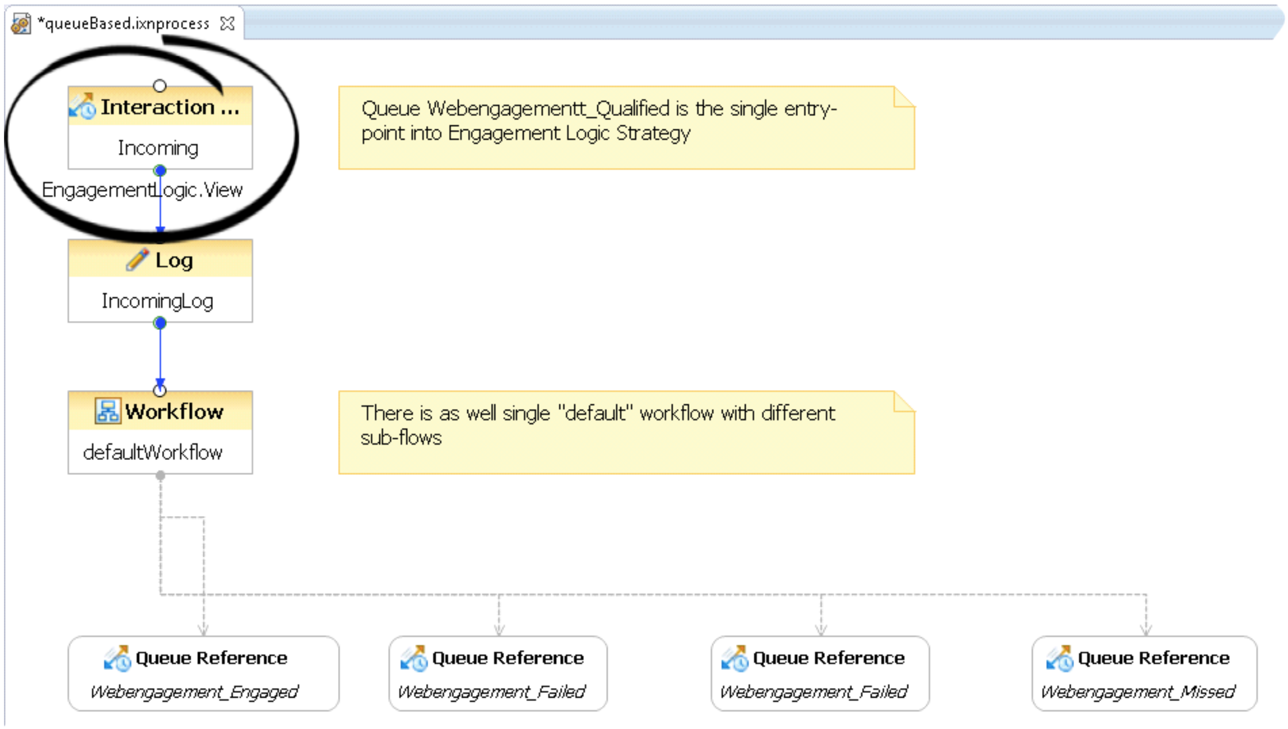
type filter text

Variable Name	Value	Description
Project Variables		
serverURL		URL of GPE Server which
serverURLSecure		Secure URL of GPE Serve
maxEngagementAttempts	3	Limit of engagement attem
maxSimultaneousEngagemen	1	Max allowed count of sim
attemptsBeforeFail	5	Maximal count of attemp

Now let's take a look at **queueBased.ixnprocess**. Select it in the Package Explorer:


Java - WebEngagement_EngagementLogic/Interaction Processes/queueBased.ixnprocess - Eclipse
File Edit Diagram Navigate Search Project Configuration Server Run Window Help
Package Explorer
WebEngagement_EngagementLogic
archive
db
include
Interaction Processes
clean.ixnprocess
queueBased.ixnprocess
META-INF
Reports
Scripts
queueBased.ixnprocess

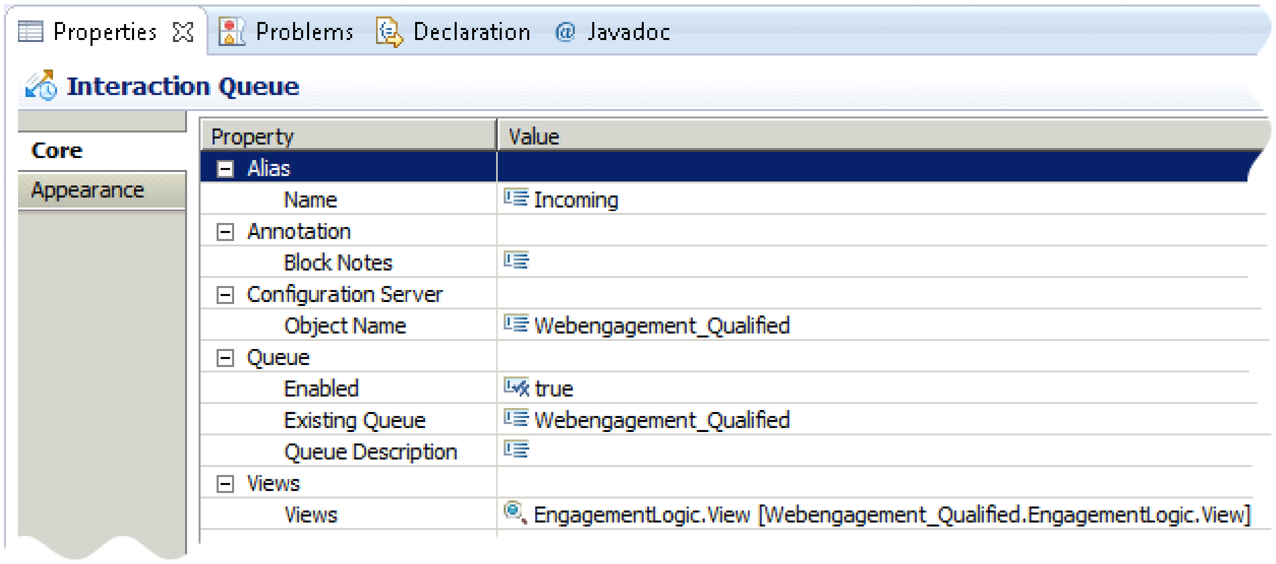
The entry point Interaction Queue (Webengagement_Qualified) is shown here:



Queue Webengagementt_Qualified is the single entry-point into Engagement Logic Strategy

There is as well single "default" workflow with different sub-flows

And its properties are here:



Property	Value
⊟ Alias	
Name	Incoming
⊟ Annotation	
Block Notes	
⊟ Configuration Server	
Object Name	Webengagement_Qualified
⊟ Queue	
Enabled	true
Existing Queue	Webengagement_Qualified
Queue Description	
⊟ Views	
Views	EngagementLogic.View [Webengagement_Qualified.EngagementLogic.View]

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.

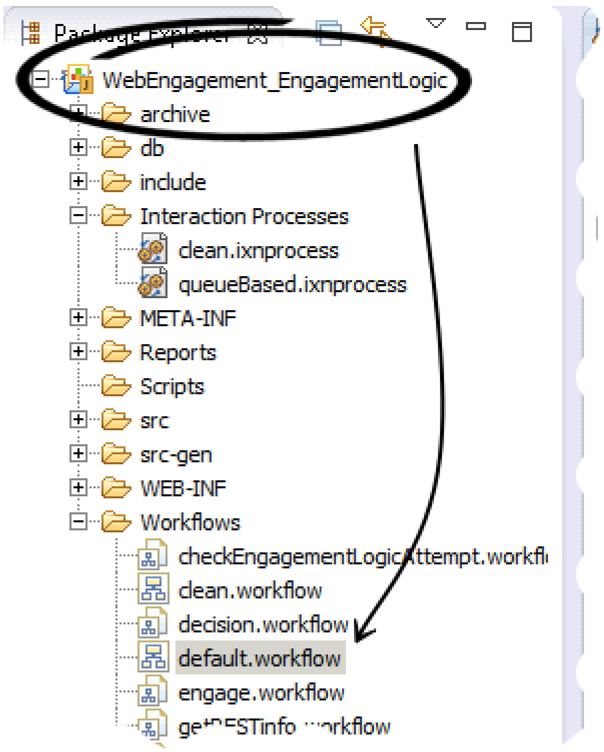
Preventing Interaction Termination into Sub-flows

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.

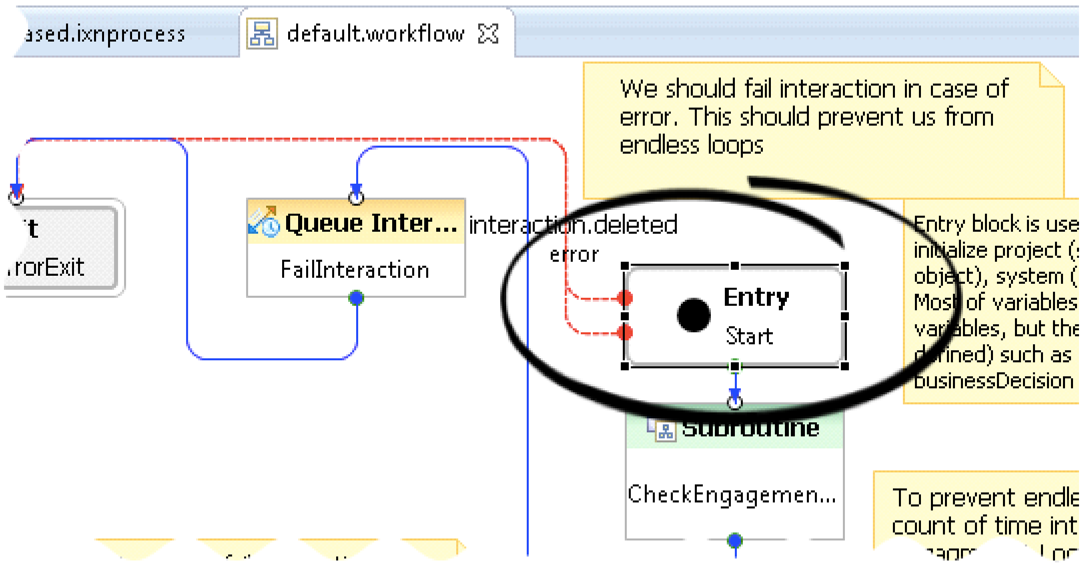
Note: Out-of-the-box Engagement Logic strategies already have the correct specified value (`0`) for the **system.TerminateIxnOnExit** variable.

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

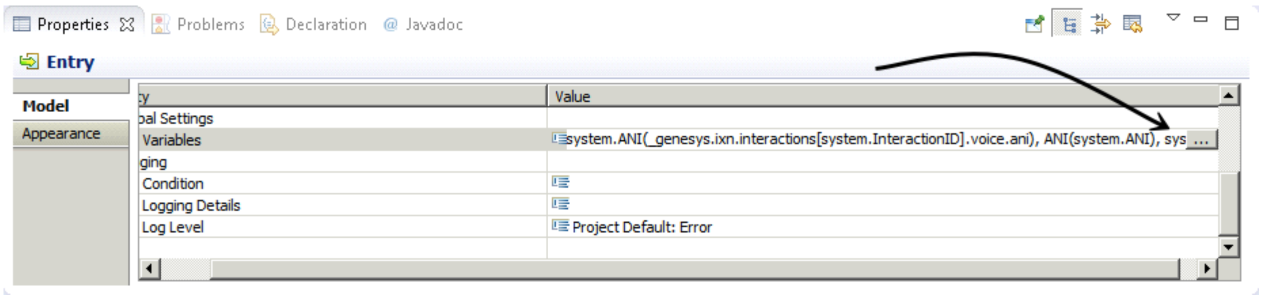
1. Open the workflow diagram in Composer (note that in the images, it is shown as **default.workflow**).



2. Select the **Entry** block.

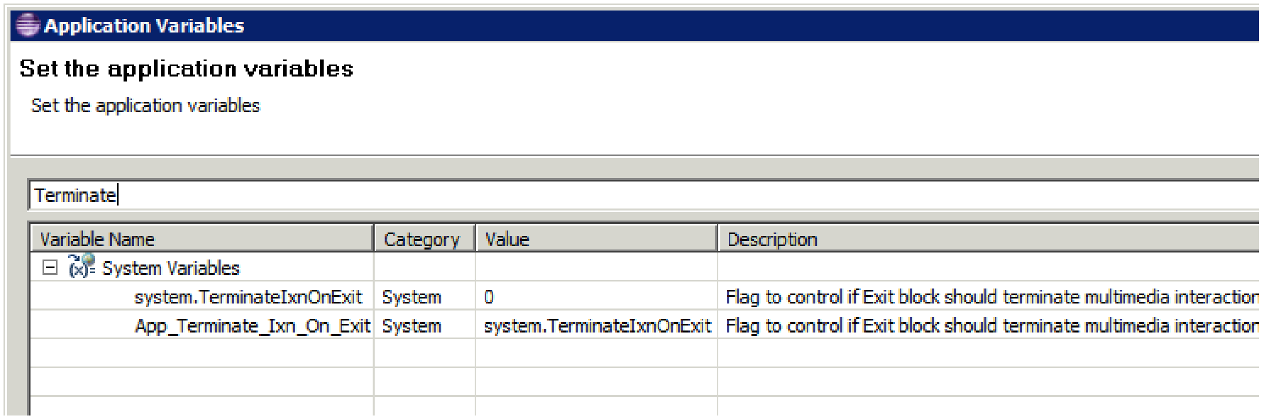


3. Open the properties of this block and access the **Global Settings > Variables**.



y	Value
bal Settings	
Variables	system.ANI(_genesys.ixn.interactions[system.InteractionID].voice.ani), ANI(system.ANI), sys
ging	
Condition	
Logging Details	
Log Level	Project Default: Error

4. Locate the variable **system.TerminateIxnOnExit**. In this case, we have filtered the variables so only those that contain the string `Terminate` are showing. Set the value to `0`.



Terminate

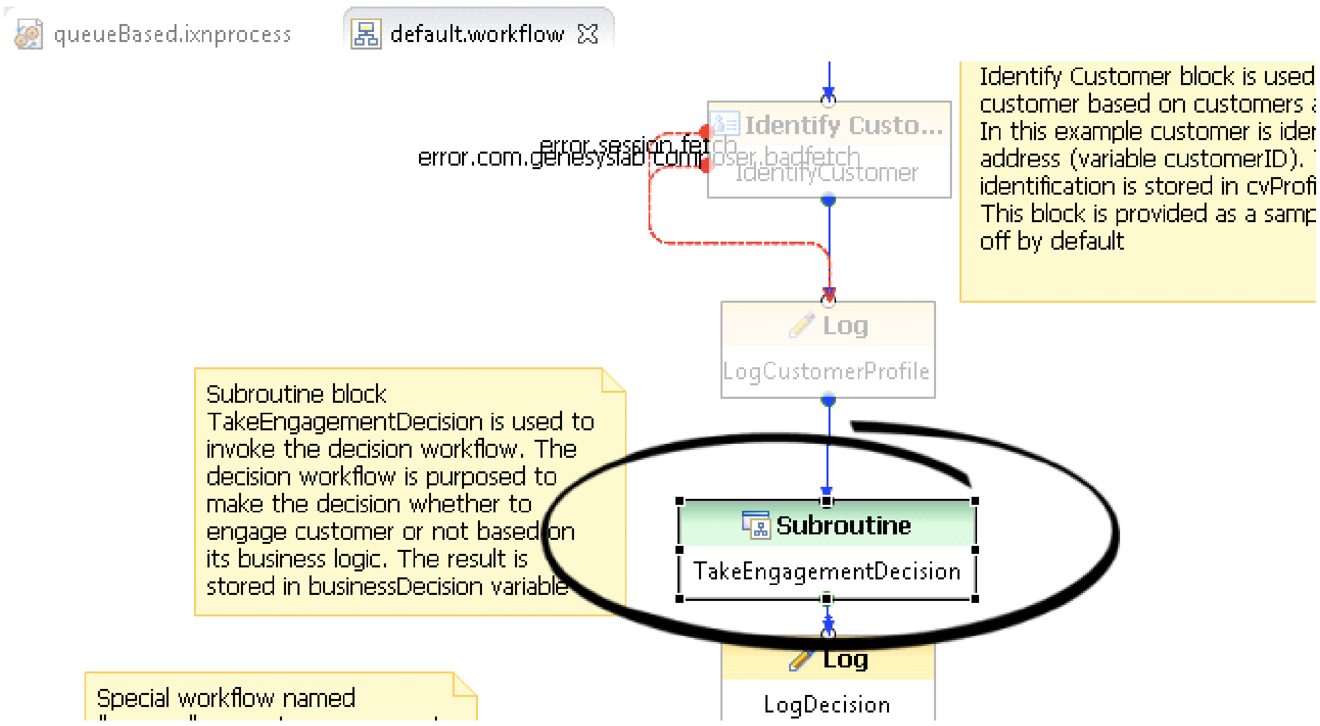
Variable Name	Category	Value	Description
System Variables			
system.TerminateIxnOnExit	System	0	Flag to control if Exit block should terminate multimedia interaction
App_Terminate_Ixn_On_Exit	System	system.TerminateIxnOnExit	Flag to control if Exit block should terminate multimedia interaction

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

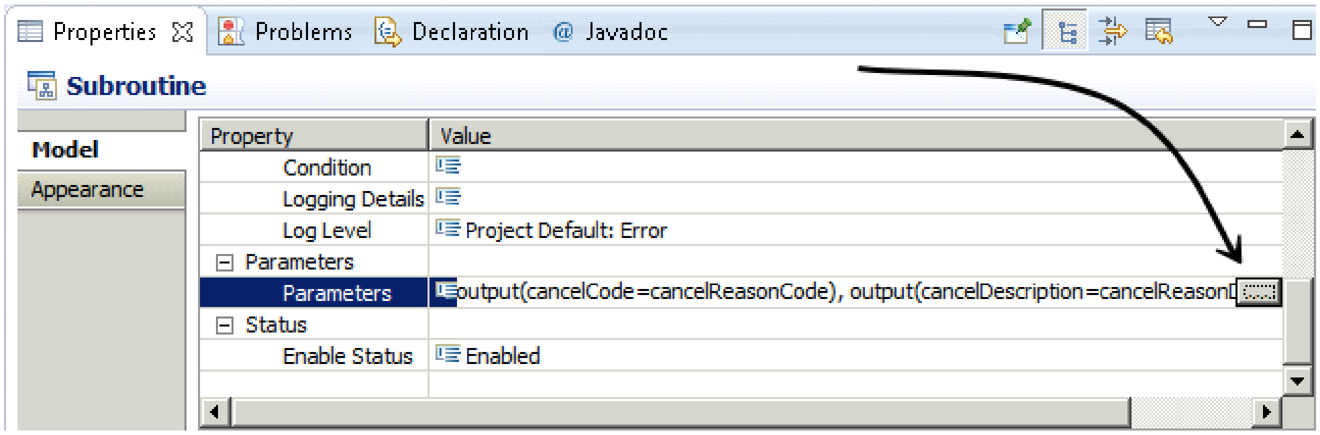
One of the most important features of the Engagement Logic is its ability to access User Data from **webengagement** interactions. This data is populated by the Web Engagement Server and includes, among other things, information provided by a pacing algorithm.

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 following example, we show the **TakeEngagementDecision** subroutine:

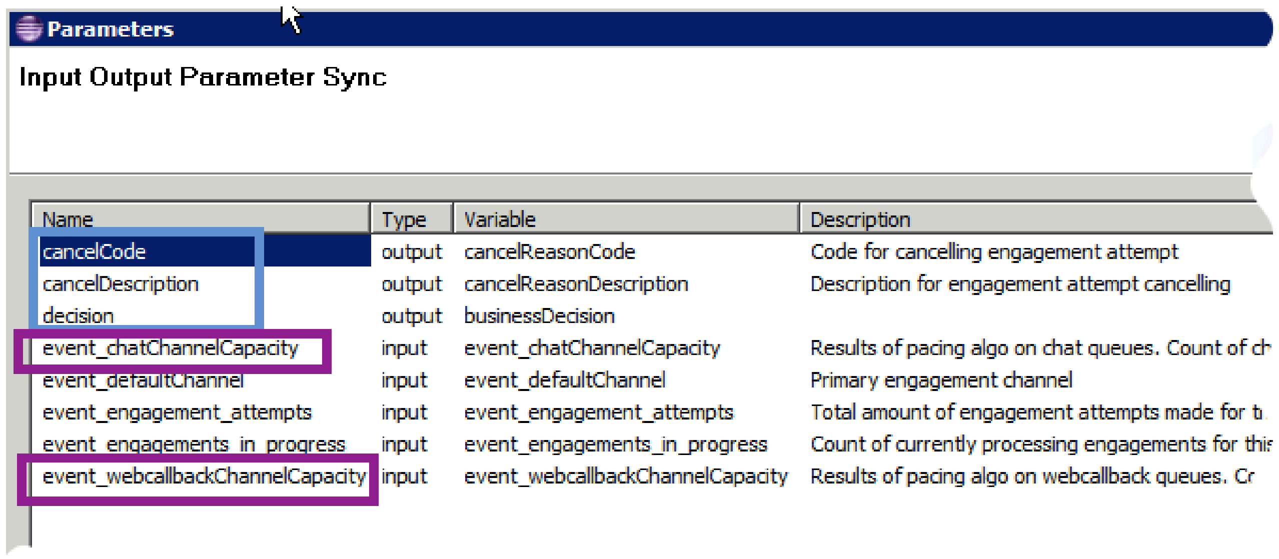

queueBased.ixnprocess
default.workflow
Identify Custo...
IdentifyCustomer
error.session.fetch
error.com.genesyslab.composer.badfetch
Identify Customer block is used
customer based on customers
In this example customer is ider
address (variable customerID).
identification is stored in cvProfi
This block is provided as a samp
off by default
Log
LogCustomerProfile
Subroutine block TakeEngagementDecision is used to invoke the decision workflow. The decision workflow is purposed to make the decision whether to engage customer or not based on its business logic. The result is stored in businessDecision variable
Subroutine
TakeEngagementDecision
Log
LogDecision
Special workflow named

Then, you can see its parameters, which are displayed in a Composer window below the workflow diagram:



Property	Value
Condition	
Logging Details	
Log Level	Project Default: Error
Parameters	
Parameters	output(cancelCode=cancelReasonCode), output(cancelDescription=cancelReasonD
Status	
Enable Status	Enabled

Let's consider the parameters we are passing into **decision.workflow**, including **event_chatChannelCapacity**, as well as the parameters we are receiving from the workflow, including,**cancelCode**, **cancelDescription** and **decision**:



Input Output Parameter Sync

Name	Type	Variable	Description
cancelCode	output	cancelReasonCode	Code for cancelling engagement attempt
cancelDescription	output	cancelReasonDescription	Description for engagement attempt cancelling
decision	output	businessDecision	
event_chatChannelCapacity	input	event_chatChannelCapacity	Results of pacing algo on chat queues. Count of ch
event_defaultChannel	input	event_defaultChannel	Primary engagement channel
event_engagement_attempts	input	event_engagement_attempts	Total amount of engagement attempts made for t
event_engagements_in_progress	input	event_engagements_in_progress	Count of currently processing engagements for thi
event_webcallbackChannelCapacity	input	event_webcallbackChannelCapacity	Results of pacing algo on webcallback queues. Cr

Attached Data in Web Engagement 8.5

As specified in the following tables, Genesys Web Engagement 8.5 supports key-value pair–based user data that is usable by Genesys Reporting.

Mandatory Actionable Event Fields

Key	Contents	Description
HotLead_eventID	UUID	eventID obtained from Actionable event
HotLead_eventName	String	Actionable event name.
HotLead_visitID	UUID	visitID obtained from Actionable event
HotLead_globalVisitID	UUID	globalVisitID obtained from Actionable event
HotLead_pageID	String	browserPageID obtained from Actionable event
HotLead_url	String	url obtained from Actionable event
HotLead_languageCode	String	languageCode obtained from Actionable event
HotLead_timestamp	long	timestamp obtained from Actionable event
HotLead_category	String	category obtained from Actionable event
HotLead_rule	String	rule obtained from Actionable event

Web Engagement Server Data

Key	Type	Description
HotLead_engagementID	UUID	ID of Engagement Profile associated with **webengagement** interaction
HotLead_engagementAttempts	int	Count of engagement attempts (accepted and rejected) that happened already on this visit
HotLead_engagementsInProgress	int	Count of currently active engagement attempts
pacing_chatCapacity	int	Actual capacity of chat channel, predicted by pacing
pacing	String	JSON object, which includes detailed group-based pacing information

Optional Fields

Key	Type	Description
HotLead_<customFieldName>	String	Field with name <customFieldName>, obtained from data object of **actionable** event. List of fields should be specified in the option eventType.ACTIONABLE ([userData] section) For example: 1) Actionable event has data fields "myCustomField" and "myAnotherCustomField": "data": {"myCustomField": "SomeValue", "myAnotherCustomField": "SomeAnotherValue"} 2) eventType.ACTIONABLE has value "myCustomField" GWE 8.5 will attach to the User Data only the following pair: "HotLead_myCustomField": "SomeValue"
VisitStarted_<customFieldName>	String	Field with name <customFieldName>, obtained from data object of **VisitStarted** event. List of fields should be specified in the option eventName.VisitStarted ([userData] section) The following keys are available: "**userAgent**",

Key	Type	Description
		"**screenResolution**", "**language**", "**timezoneOffset**" In OOB template option **eventName.VisitStarted** has value "**timezoneOffset**" Correspondingly, GWE 8.5 will attach to the User Data the following pair: "**VisitStarted_timezoneOffset**": 25200000 (value will depend on visitor's timezone)
SignIn_<customFieldName>	String	Field with name <customFieldName>, obtained from data object of **SignIn** event. List of fields should be specified in the option eventName.SignIn ([userData] section) List of available keys depends on customer's workflow
UserInfo_<customFieldName>	String	Field with name <customFieldName>, obtained from data object of **UserInfo** event. List of fields should be specified in the option eventName.UserInfo ([userData] section) List of available keys depends on customer's workflow

Engagement Policy (Decision Workflow)

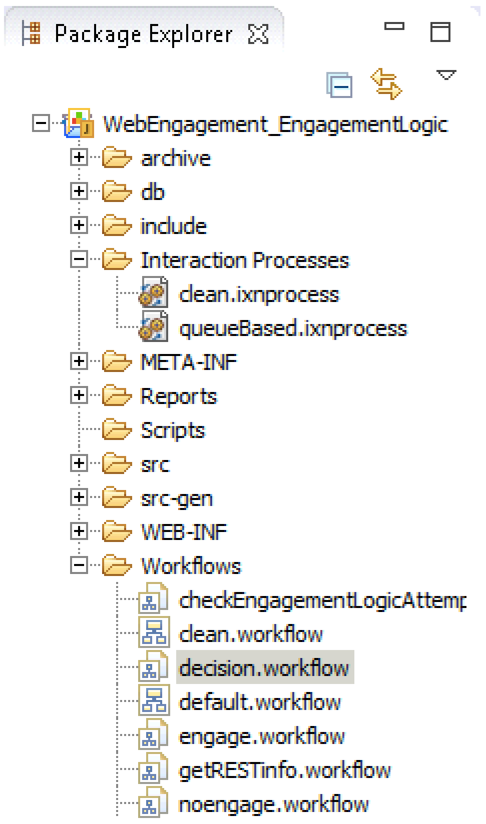
Engagement policy is the other name of decision workflow.

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

Count of Engagement Attempts

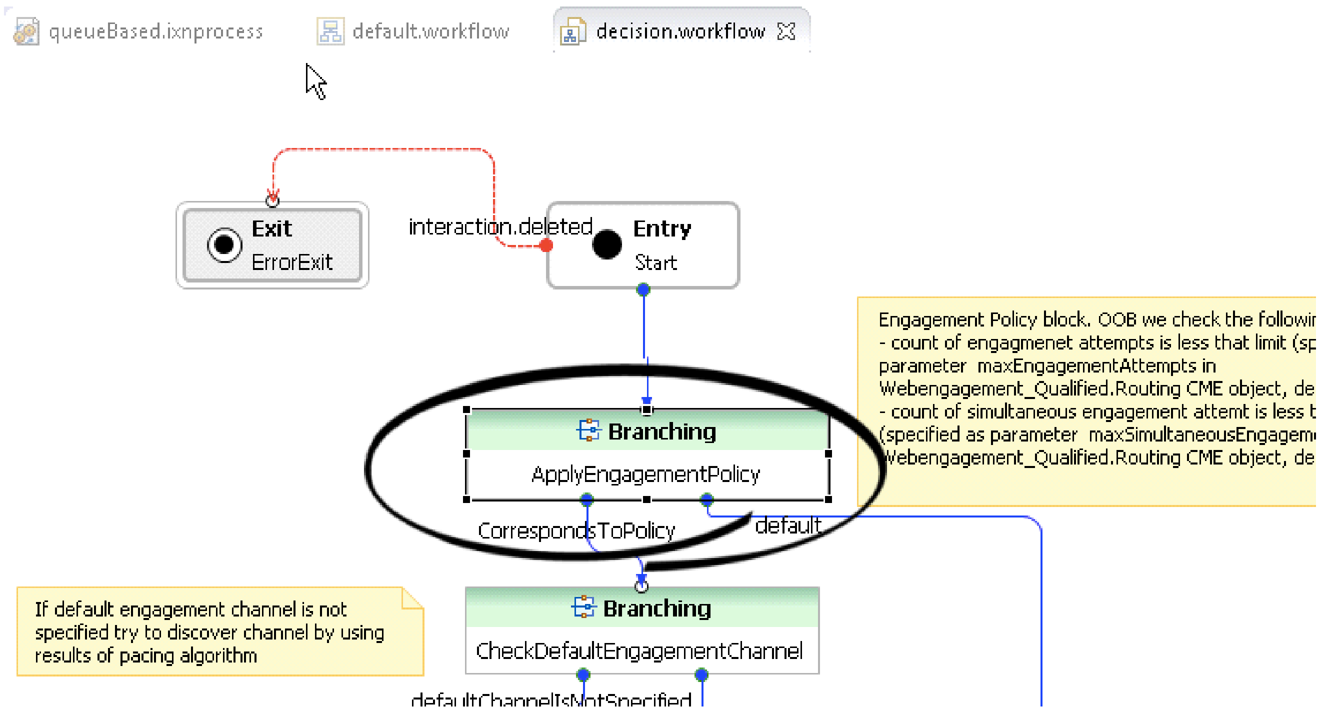
Check the count of engagement attempts already proposed to the current visitor.

To see where this check is executed open **decision.workflow**:

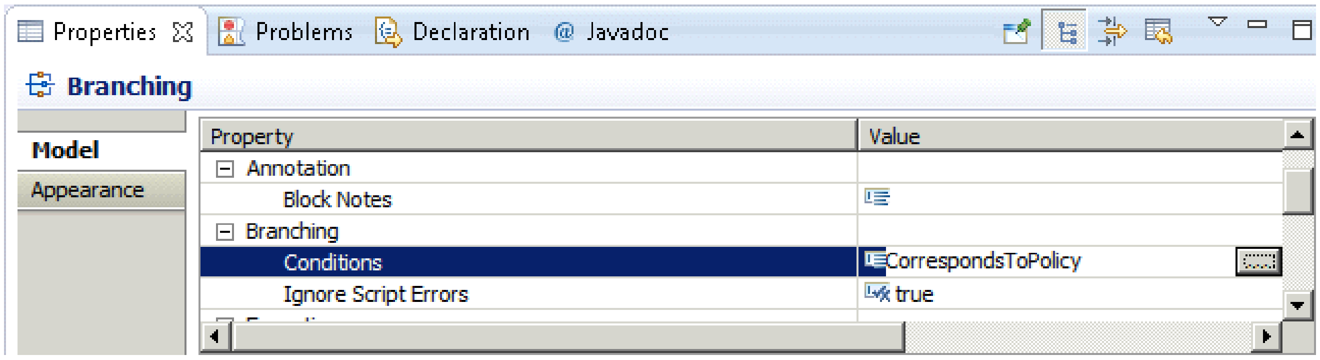


```
WebEngagement_EngagementLogic
  archive
  db
  include
  Interaction Processes
    clean.ixnprocess
    queueBased.ixnprocess
  META-INF
  Reports
  Scripts
  src
  src-gen
  WEB-INF
  Workflows
    checkEngagementLogicAttemp
    clean.workflow
    decision.workflow
    default.workflow
    engage.workflow
    getRESTinfo.workflow
    noengage.workflow
```

Looking at the workflow, you can select the **ApplyEngagementPolicy** block:

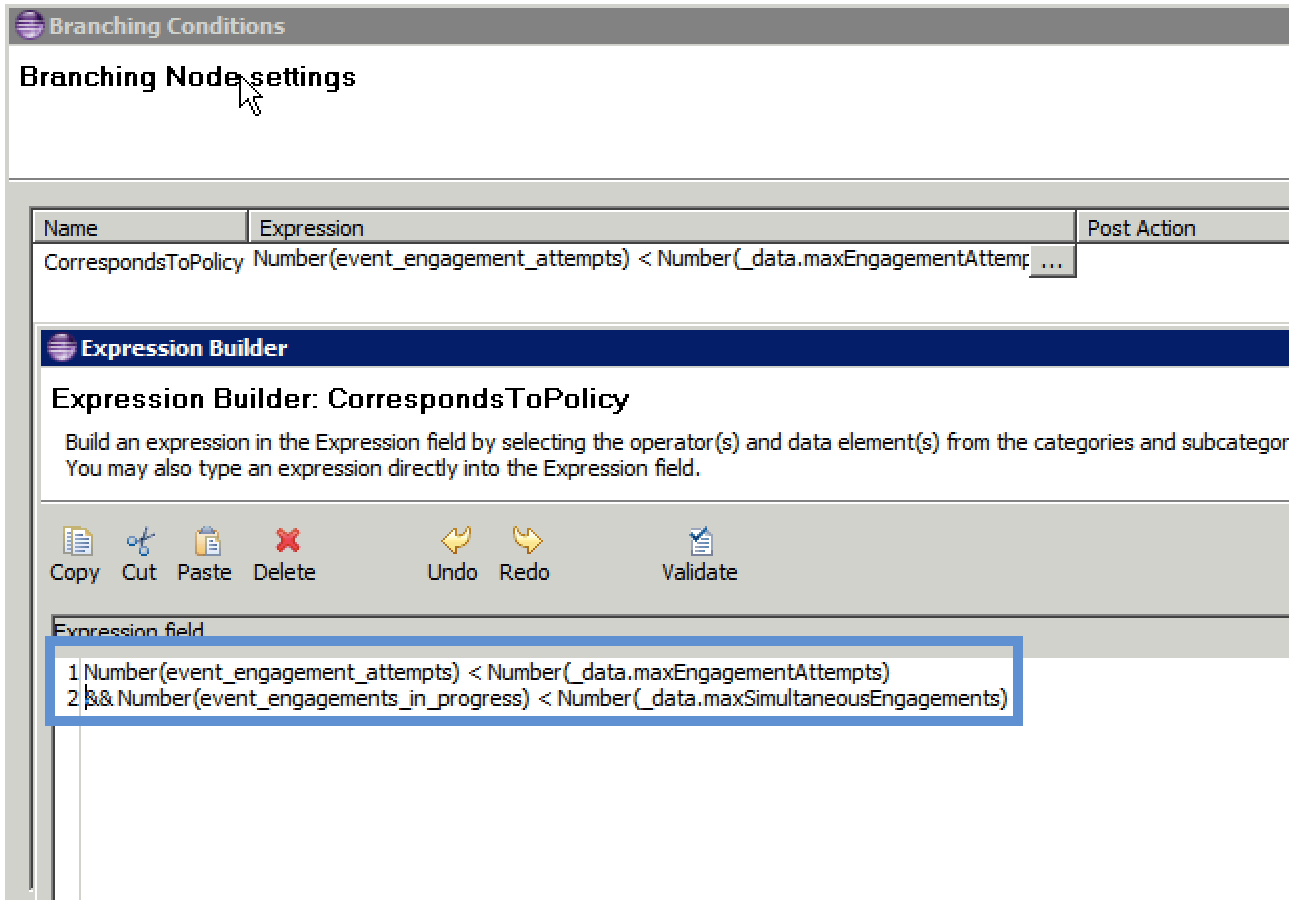

queueBased.ixnprocess
default.workflow
decision.workflow
Exit
ErrorExit
interaction.deleted
Entry
Start
Engagement Policy block. OOB we check the followi
- count of engagmenet attempts is less that limit (sp
parameter maxEngagementAttempts in
Webengagement_Qualified.Routing CME object, de
- count of simultaneous engagement attemt is less t
(specified as parameter maxSimultaneousEngagem
Webengagement_Qualified.Routing CME object, de
Branching
ApplyEngagementPolicy
CorrespondsToPolicy
default
If default engagement channel is not specified try to discover channel by using results of pacing algorithm
Branching
CheckDefaultEngagementChannel
defaultChannelIsNotSpecified

In the properties for this block, select **Branching > Conditions** and open **CorrespondsToPolicy**:



Property	Value
Annotation	
Block Notes	
Branching	
Conditions	CorrespondsToPolicy
Ignore Script Errors	true

CorrespondsToPolicy is an expression that uses application parameters from the **Webengagement_Qualified.Routing** script object to determine how many engagement attempts should be proposed for a particular visitor. **Note:** Engagement attempts in the current visit that were closed with a timeout disposition code will not be taken into account, as there is no guarantee whether the visitor has seen them. For example, the invitation may appear on a non-active browser tab or window.



CorrespondsToPolicy Number(event_engagement_attempts) < Number(_data.maxEngagementAttemp

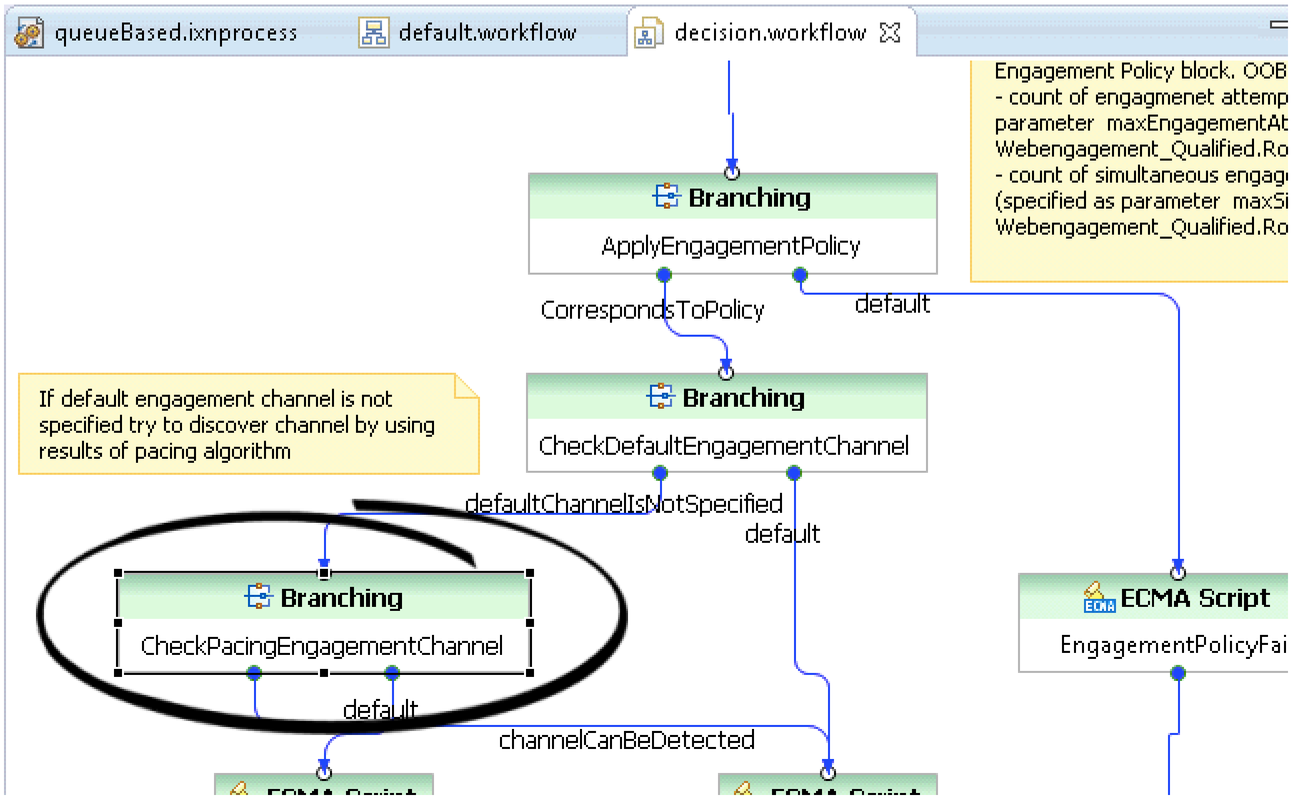
Expression Builder: CorrespondsToPolicy

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategor
You may also type an expression directly into the Expression field.

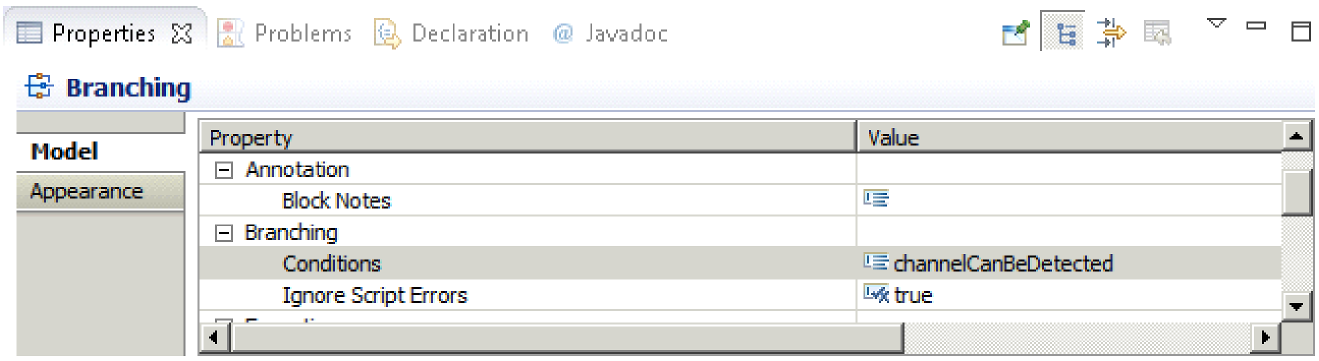
```
Number(event_engagement_attempts) < Number(_data.maxEngagementAttempts)
&& Number(event_engagements_in_progress) < Number(_data.maxSimultaneousEngagements)
```

Pacing Information

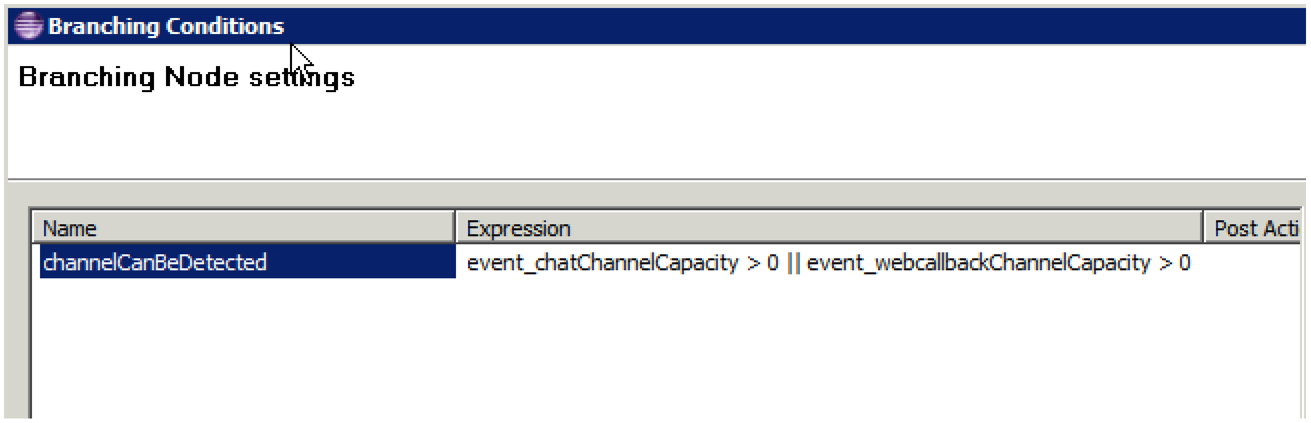
Check pacing information. This is executed inside of the **CheckPacingEngagementChannel** block:


queueBased.ixnprocess
default.workflow
decision.workflow
Engagement Policy block. OOB
- count of engagmenet attemp
parameter maxEngagementAt
Webengagement_Qualified.Ro
- count of simultaneous engag
(specified as parameter maxSi
Webengagement_Qualified.Ro
Branching
ApplyEngagementPolicy
CorrespondsToPolicy
default
If default engagement channel is not specified try to discover channel by using results of pacing algorithm
Branching
CheckDefaultEngagementChannel
defaultChannelIsNotSpecified
default
Branching
CheckPacingEngagementChannel
ECMA Script
EngagementPolicyFai
default
channelCanBeDetected

Note: The out-of-the-box strategy operates only on general information obtained from the pacing algorithm: in particular, the **event_chatChannelCapacity** variable, which is passed from **default.workflow**, contains the accumulated count of interactions that can be triggered at a particular moment. You can also pass more detailed information provided by the pacing algorithm into the decision workflow and build a more sophisticated decision maker. The images below show the general idea: do **not** engage the visitor if the count of available "interactions to produce" is `0` for both channels:



Property	Value
⊟ Annotation	
Block Notes	
⊟ Branching	
Conditions	channelCanBeDetected
Ignore Script Errors	true



Branching Node settings

Name	Expression	Post Acti
channelCanBeDetected	event_chatChannelCapacity > 0 || event_webcallbackChannelCapacity > 0	

Obtaining Data from the GWE Cassandra Database through REST Requests

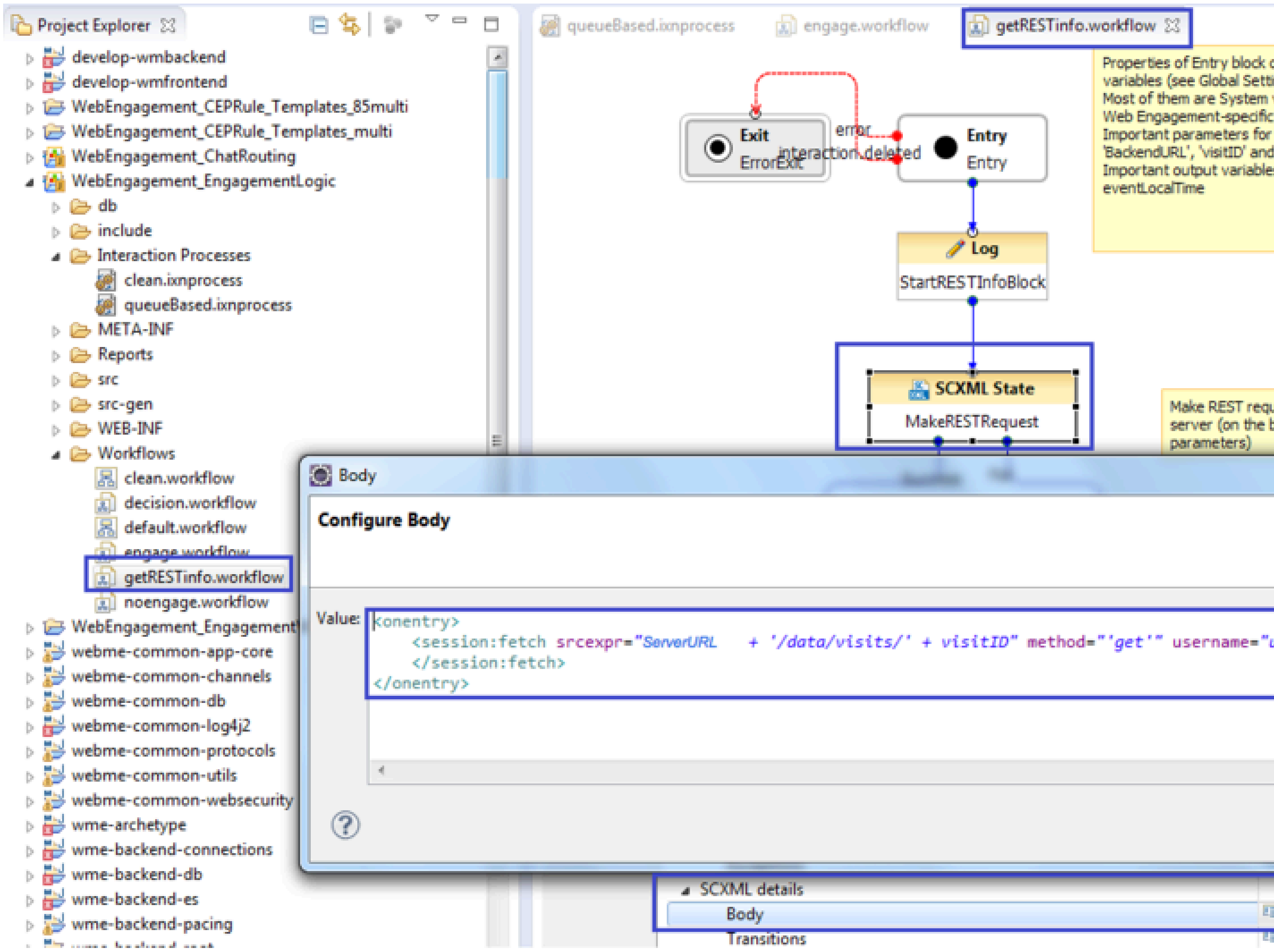
Requesting data from Web Engagement Server through the REST

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. **Note:** the **SCXML State** block that is used to demonstrate these concepts is disabled by default in Web Engagement 8.5. It has only been retained as a sample, because the GWE 8.5 server provides related information as a part of the User Data in the **webengagement** open media interaction.

Consider how Engagement Strategy does this task.

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

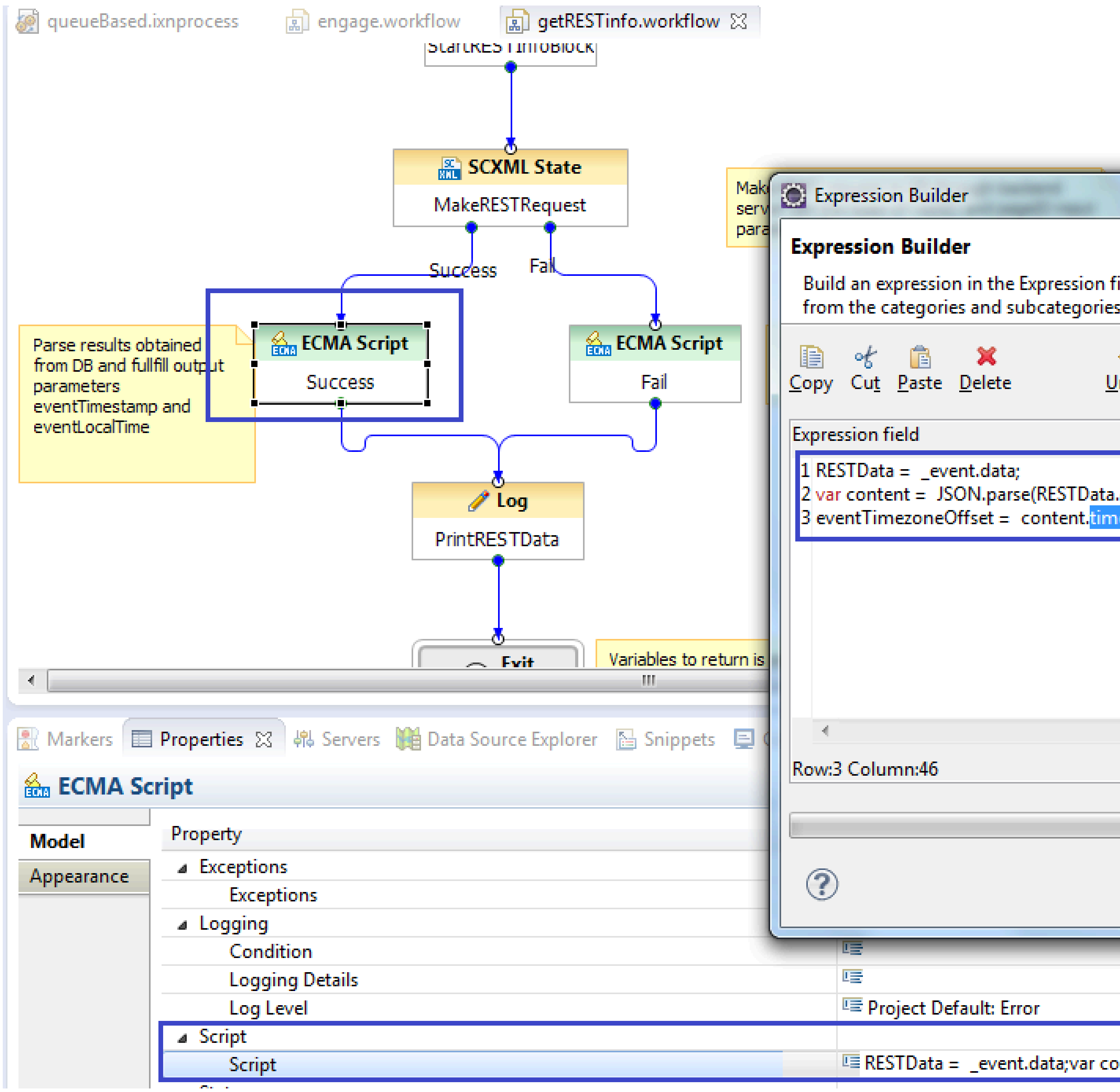


```
<onentry>
    <session:fetch srcexpr="ServerURL    + '/data/visits/' + visitID" method="'get'" username="u
    </session:fetch>
</onentry>
```

Use the State block to make REST requests

Note: The **ServerURL** 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:



```
1 RESTData = _event.data;
2 var content = JSON.parse(RESTData.
3 eventTimezoneOffset = content.tim
```

RESTData = _event.data;var co

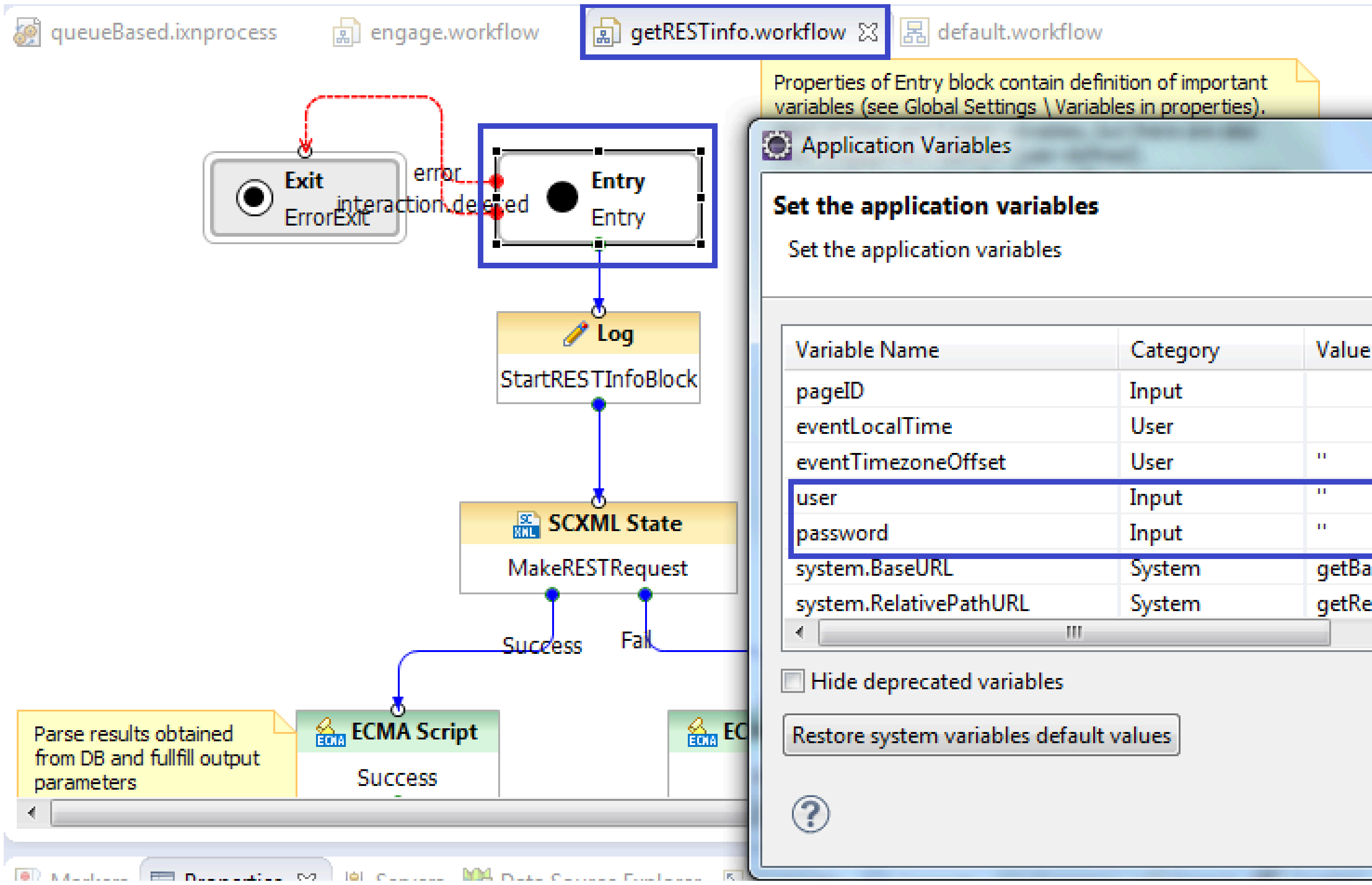
Parse the response to the REST request

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 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:



Properties of Entry block contain definition of important variables (see Global Settings \ Variables in properties).

Parse results obtained from DB and fullfill output parameters

Application Variables

Set the application variables

Set the application variables

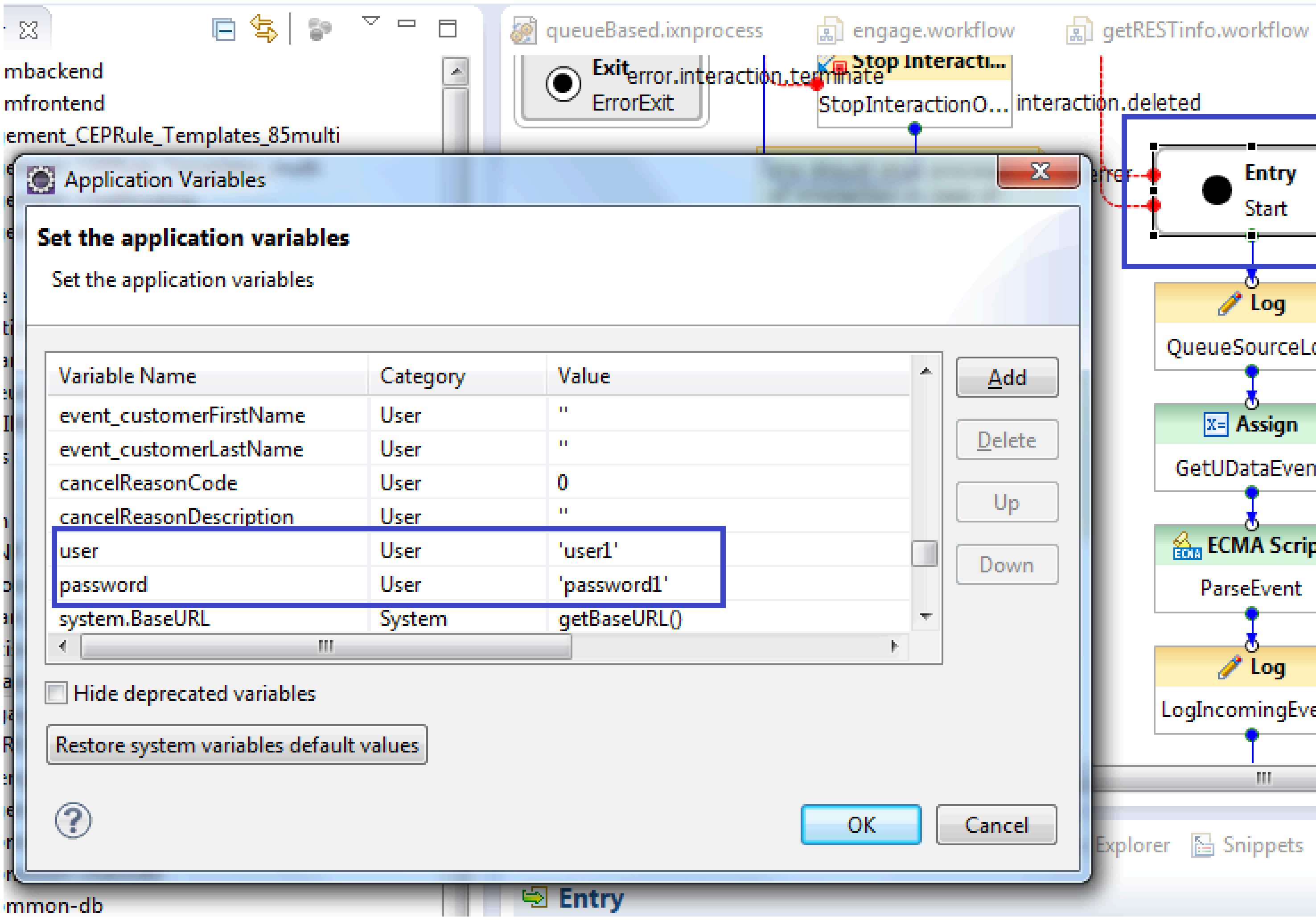
Variable Name	Category	Value
pageID	Input	
eventLocalTime	User	
eventTimezoneOffset	User	''
user	Input	''
password	Input	''
system.BaseURL	System	getBa
system.RelativePathURL	System	getRe

☐ Hide deprecated variables

Restore system variables default values

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**:



Set the application variables

Set the application variables

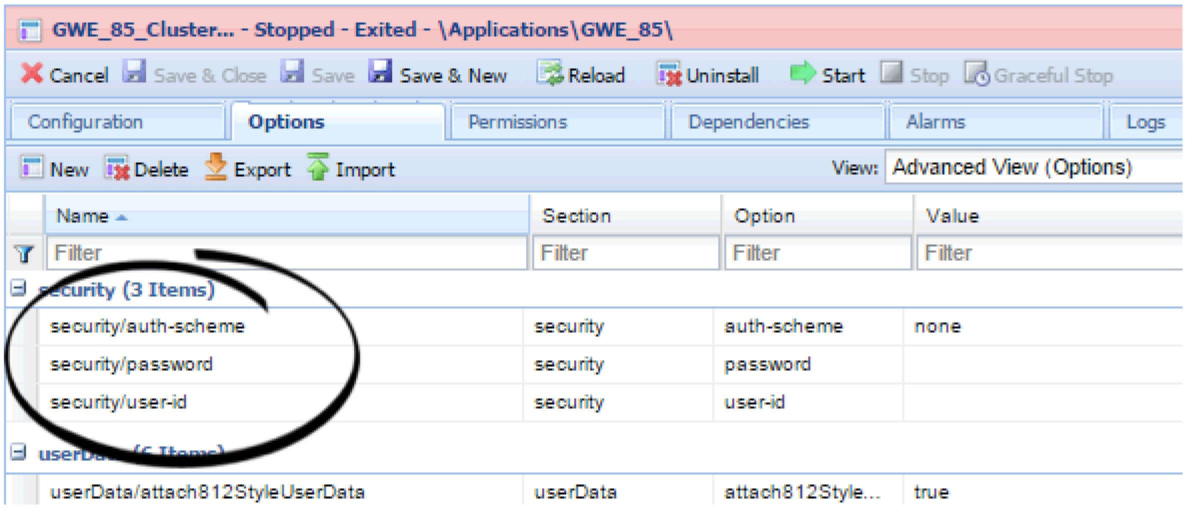
Variable Name	Category	Value
event_customerFirstName	User	''
event_customerLastName	User	''
cancelReasonCode	User	0
cancelReasonDescription	User	''
user	User	'user1'
password	User	'password1'
system.BaseURL	System	getBaseURL()

☐ Hide deprecated variables

Restore system variables default values

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

You must check that these credentials are compliant with the credentials specified in the security section of the Web Engagement Cluster or Web Engagement Server options:


GWE_85_Cluster... - Stopped - Exited - \Applications\GWE_85\

Name	Section	Option	Value
security (3 Items)			
security/auth-scheme	security	auth-scheme	none
security/password	security	password	
security/user-id	security	user-id	
userData (6 Items)			
userData/attach812StyleUserData	userData	attach812Style...	true

The username and password are specified in the security section

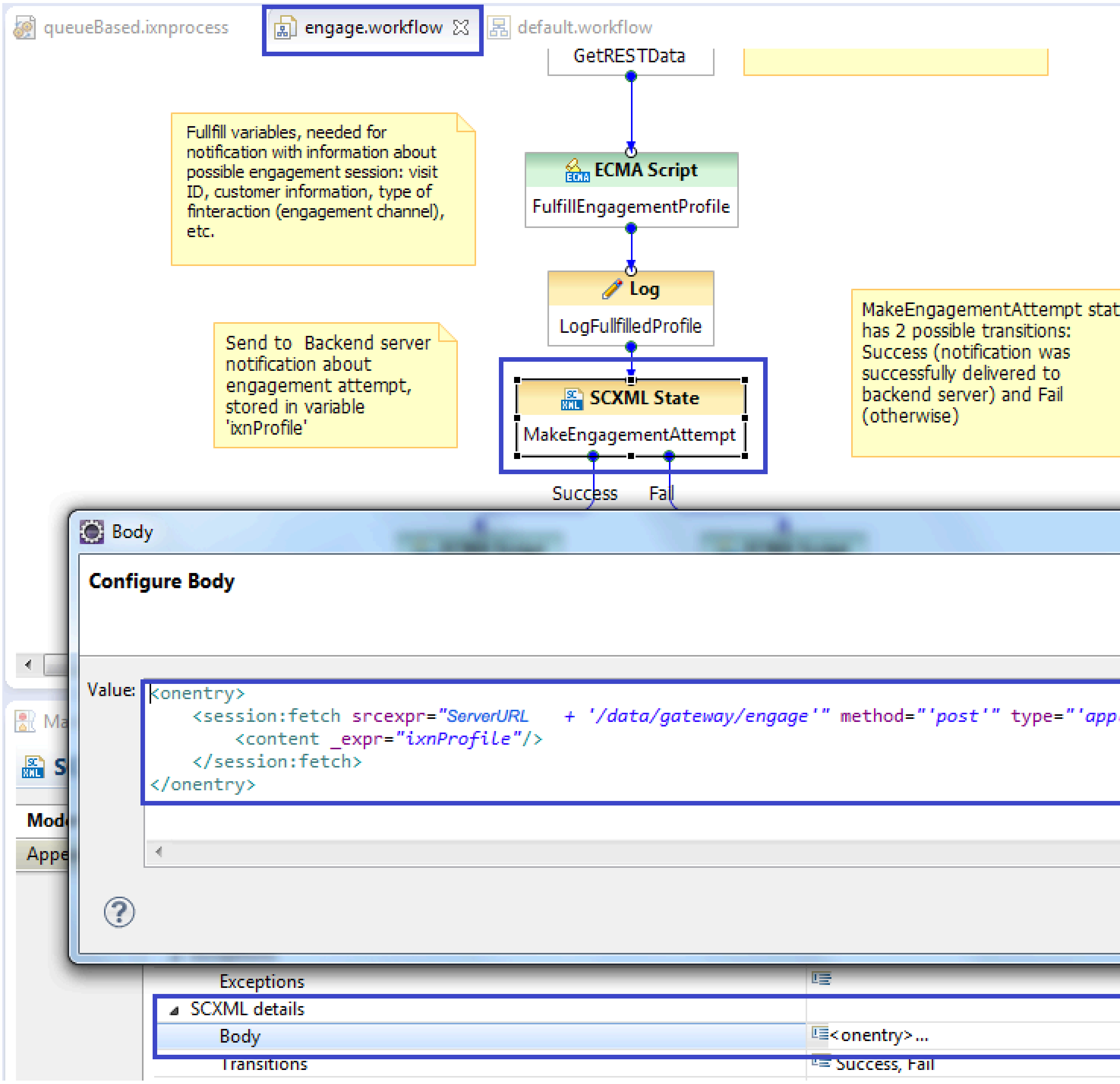
See Configuring Authentication for details.

Start Engagement as a Result of the Engagement Logic Strategy

Sending the "start engage" Request to the Web Engagement Server

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

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


queueBased.ixnprocess
engage.workflow
default.workflow
GetRESTData
Fullfill variables, needed for notification with information about possible engagement session: visit ID, customer information, type of finteraction (engagement channel), etc.
ECMA Script
FulfillEngagementProfile
Log
LogFullfilledProfile
Send to Backend server notification about engagement attempt, stored in variable 'ixnProfile'
SCXML State
MakeEngagementAttempt
MakeEngagementAttempt stat
has 2 possible transitions: Success (notification was successfully delivered to backend server) and Fail (otherwise)
Success
Fail
Body
Configure Body
Value:
Exceptions
SCXML details
Body
<onentry>...
Transitions
Success, Fail

```
<onentry>
    <session:fetch srcexpr="ServerURL   + '/data/gateway/engage'" method="'post'" type="'app
        <content _expr="ixnProfile"/>
    </session:fetch>
</onentry>
```

The REST request notifies the Web Engagement 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 Web Engagement 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 = {
    'profile': engageProfile,
    'notification': notification_message
}
```

As you can see, there are two fields:

- profile — represented by the variable **engagementProfile**.
    - The 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.
- notification — represented by the variable **notification_message**.

The structure of the notification message is described in Chat Invitation Message.

Structure of the engagementProfile variable

Field name	Field contents	Description
engagementID	UUID	Auto-generated field which identifies exactly one engagement attempt
visitID	UUID	visitID of current session (obtained from HotLeadActionableEvent)
globalVisitID	UUID	globalVisitID of current session (obtained from HotLeadActionableEvent)
webengagementInteractionID	String	ID of "webengagement" OM interaction associated with this Engagement Profile
pageID	String	PageID identified specific tab in browser (obtained from HotLeadActionableEvent)
category	String	List of categories specified in HotLeadActionableEvent

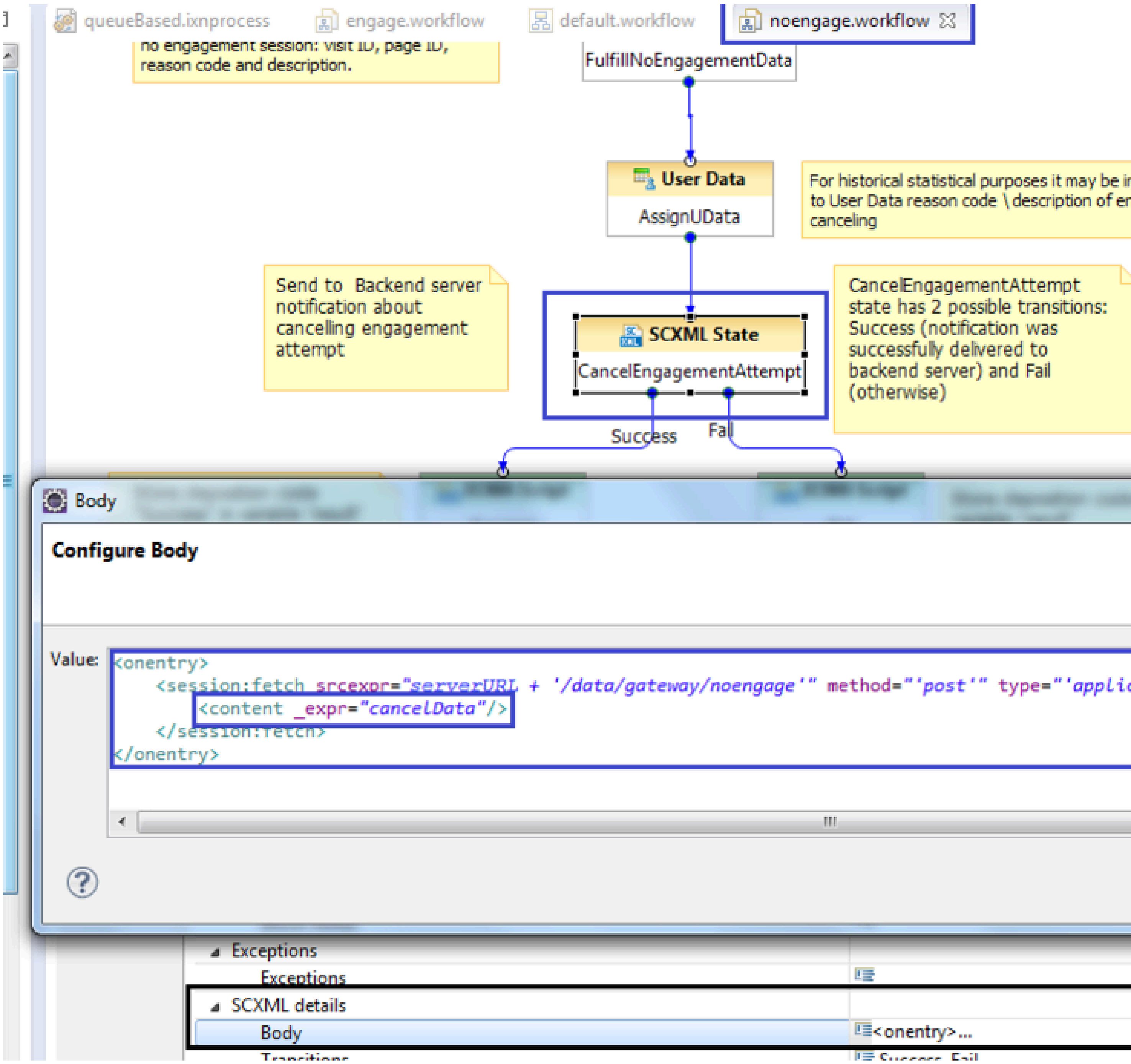
Field name	Field contents	Description
rule	String	Name of rule, which triggered this HotLeadActionableEvent
userID	String	String, which allows to identify authorized and recognized visitors For anonymous users it will be null
userState	String	State of current visit: Anonymous, Recognized or Authorized
firstName	String	First name of non-anonymous user
lastName	String	Last name of non-anonymous user
userData	String	JSON string which represents User Data, collected on webengagement OM interaction before submit and in the Engagement Logic strategy

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

Cancelling Engagement as a Result of the Engagement Logic Strategy

Sending "cancel engagement" to Web Engagement 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 Web Engagement Server:


queueBased.ixnprocess
engage.workflow
default.workflow
noengage.workflow
no engagement session: visit ID, page ID, reason code and description.
FulfillNoEngagementData
User Data
AssignUData
For historical statistical purposes it may be i
to User Data reason code \ description of e
canceling
Send to Backend server notification about cancelling engagement attempt
SCXML State
CancelEngagementAttempt
CancelEngagementAttempt state has 2 possible transitions: Success (notification was successfully delivered to backend server) and Fail (otherwise)
Success
Fail
Body
Configure Body
Value:
Exceptions
Exceptions
SCXML details
Body
<onentry>...

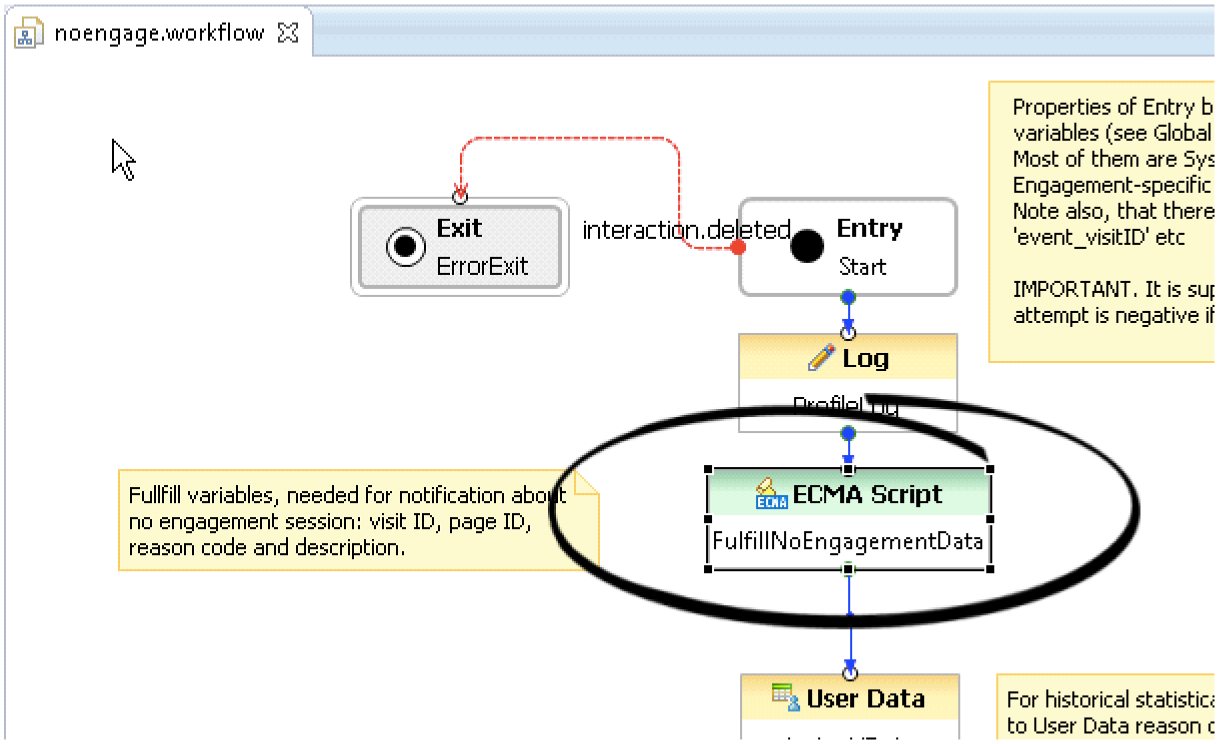
```
<onentry>
    <session:fetch srcexpr="serverURL + '/data/gateway/noengage'" method="'post'" type="'applic
        <content _expr="cancelData"/>
    </session:fetch>
</onentry>
```

The REST request cancels the engagement

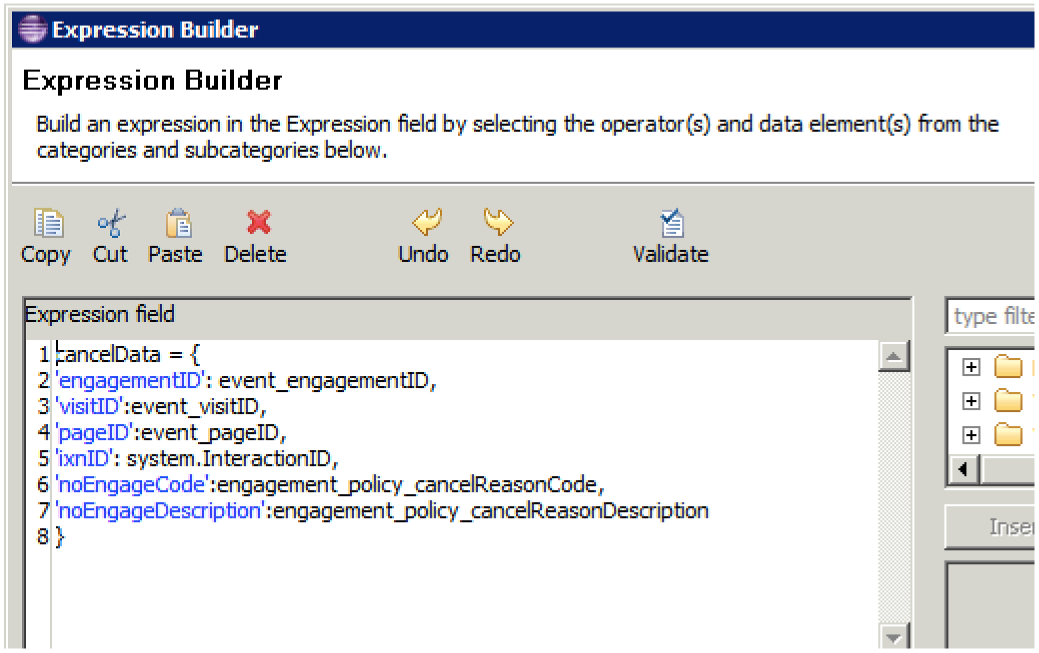
Security (authentication) aspects are the same as described in the **getRESTInfo.workflow**.

Fulfilling "no engage" Data

no engage data is available in the script properties of the **FulfillNoEngagementData** block:



It contains six mandatory fields:



Expression Builder

Build an expression in the Expression field by selecting the operator(s) and data element(s) from the categories and subcategories below.

```
cancelData = {
'engagementID': event_engagementID,
'visitID':event_visitID,
'pageID':event_pageID,
'ixnID': system.InteractionID,
'noEngageCode':engagement_policy_cancelReasonCode,
'noEngageDescription':engagement_policy_cancelReasonDescription
}
```

Cleaning Interaction Process

The cleaning process was responsible for removing stuck **webengagement** interactions. An interaction can be stuck in one of the interaction queues for various reasons. For example:

- Visitor obtained engagement invitation. This means that the **webengagement** interaction was put into the Webengagement_Accepted queue.
- Power-off appeared on visitor's host, so the answer (Accept, Reject, or Timeout) was not delivered to Genesys Web Engagement.

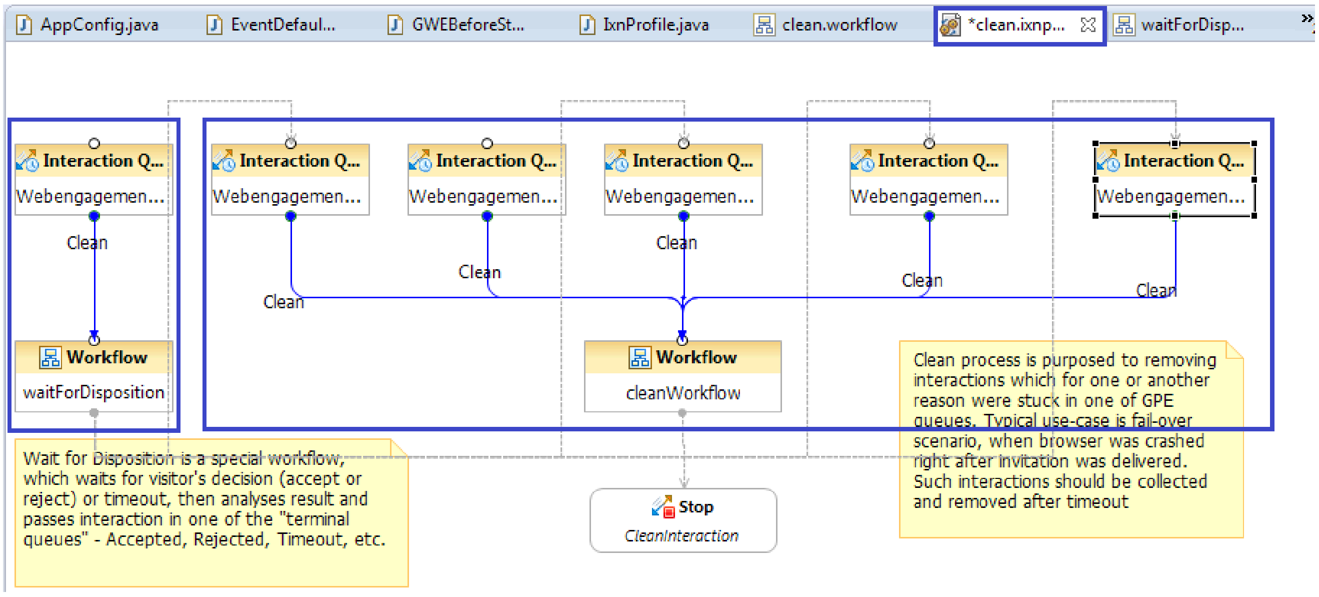
In this case, you need to define the cleaning process, which is also built on the top of ORS strategies.

The cleaning interaction process also carries out some other important functions. It is responsible not only for cleaning stuck interactions, but also for the entire life cycle of **webengagement** Open Media interactions, including these functions:

- Detecting when an interaction should be moved into a specific Interaction Queue
- Moving an interaction through the Interaction Queues
- Detecting when an interaction should be terminated
- Terminating an interaction

The Cleaning process has 6 entry points:

- Webengagement_Engaged
- Webengagement_Accepted
- Webengagement_Missed
- Webengagement_Rejected
- Webengagement_Failed
- Webengagement_Timeout



Wait for Disposition is a special workflow, which waits for visitor's decision (accept or reject) or timeout, then analyses result and passes interaction in one of the "terminal queues" - Accepted, Rejected, Timeout, etc.

Clean process is purposed to removing interactions which for one or another reason were stuck in one of GPE queues. Typical use-case is fail-over scenario, when browser was crashed right after invitation was delivered. Such interactions should be collected and removed after timeout

Note that the Webengagement_Qualified queue is no longer monitored by the cleaning process. It is only used in the main process.

The cleaning process has two workflows:

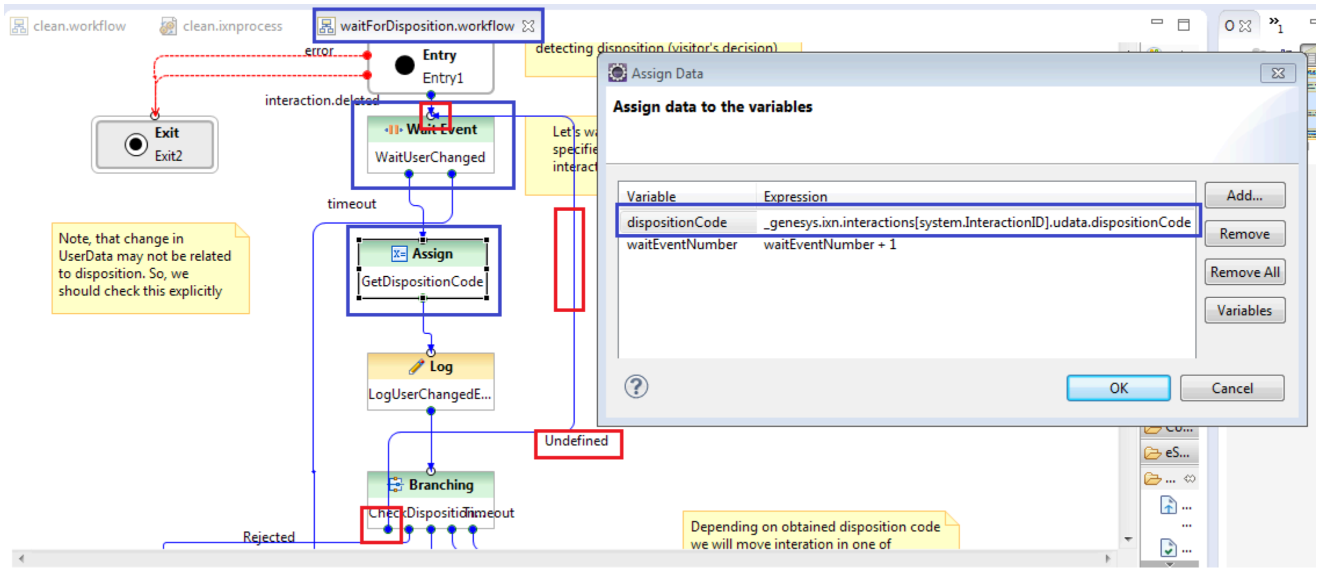
- **waitForDisposition.workflow**
- **clean.workflow**

The **waitForDisposition.workflow** only works with the Webengagement_Engaged queue, while **clean.workflow** works with all other queues and is extremely simple, as it only stops the interaction.

The "Wait for disposition" flow

This new workflow is dedicated to listening for User Data changes in **webengagement** interactions and deciding which Interaction Queue the interaction should be moved to.

The interaction's disposition code (`accept`, `reject`, and so on) will be available in User Data as a key-value pair with a key of `dispositionCode`. As soon as the `dispositionCode` key-value pair is obtained, the result will be analyzed.



Assign data to the variables

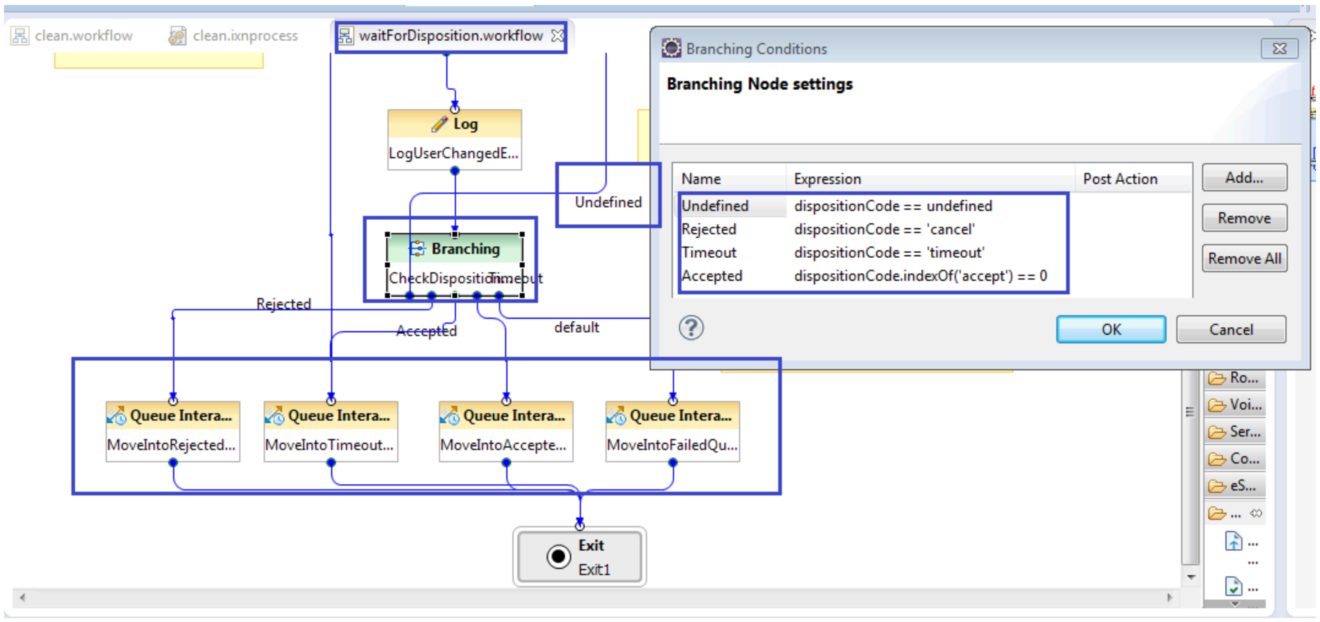
Variable	Expression
dispositionCode	_genesys.ixn.interactions[system.InteractionID].udata.dispositionCode
waitEventNumber	waitEventNumber + 1

Here are the valid values for `dispositionCode` and the queues their interactions are placed in:

Value	Description	Queue
accept	The visitor has accepted the engagement invite	Webengagement_Accepted
cancel	The visitor has cancelled the engagement invite	Webengagement_Rejected
timeout	The engagement invite has timed out	Webengagement_Timeout
pageExit	The visitor has exited the page	Webengagement_Failed

Notes

- For all other disposition code values, the associated interaction will be placed in the Webengagement_Failed queue.
- If the disposition code is not defined, the strategy will wait for the next User Data change or for a timeout.
- Disposition codes values are case-sensitive. For example, on receiving a disposition code of `Accept` (instead of `accept`) Web Engagement will place the associated interaction in the Webengagement_Failed queue
- If a timeout occurs, the interaction will be placed in the Webengagement_Timeout queue.



Branching Node settings

Name	Expression	Post Action
Undefined	dispositionCode == undefined	
Rejected	dispositionCode == 'cancel'	
Timeout	dispositionCode == 'timeout'	
Accepted	dispositionCode.indexOf('accept') == 0	

The "Cleaning" flow

The cleaning flow is quite simple: it stops the interaction. It operates with 5 terminal Interaction Queues:

- Webengagement_Accepted
- Webengagement_Missed
- Webengagement_Rejected
- Webengagement_Failed
- Webengagement_Timeout

As soon as the interaction reaches one of these queues, it will be stopped by the strategy.

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. The Web Engagement Server automatically attaches this data to the User Data of the webengagement interaction, so that it can be used in the Engagement Logic SCXML to make an engagement decision. But you can also propagate it partially, or entirely, to the chat interactions, so that it can be used in the chat routing strategies.

For example:

- Business data produced on the page provides information about language.
- This information is passed to the **webengagement** interaction as part of the User Data.
- During the execution of the Engagement Logic strategy, language information is extracted from the User Data of the **webengagement** Open Media interaction and placed into the **userData** option of the notification message.
- The notification message is processed by Genesys Widgets WebChat widget and **userData** is parsed and attached to the chat media interaction initiated by Genesys Widgets.
- The Chat Routing strategy reads language information from the User Data of the chat interaction and decides which group to route the chat interaction to.

The following are details of the described data propagation.

Attach UserData to the webengagement Interaction

All of the data contained in a data property of a triggered **HotleadIdentifiedActionableEvent** can be attached to the User Data of a **webengagement** Open Media interaction and accessed by the Engagement Logic SCXML strategy.

Fields attached to the User Data of a **webengagement** Open Media interaction are specified by the different options in the [userData] section.

Control Copying UserData to the Chat Interaction

Starting from GWE 8.5.000.38, native GWE widgets are deprecated and Genesys Widgets are now the primary integration point. This means that chat interactions are initiated by the WebChat widget. This widget parses **userData** passed in the notification message from GWE and attaches it to the newly created chat interaction.

The notification message is formed by the Engagement Logic SCXML Strategy or directly in the rules file and then passed through the GWE Server to the browser. In addition, it is possible to implicitly inject data available in the **HotleadIdentifiedActionableEvent** into the notification message. You can control how **HotleadIdentifiedActionableEvent** data is injected into the notification message by using the keysToPropagate option in the [userData] section of the Web Engagement Server application.

This option has three modes:

- Copy all data into userData of the notification message
- Do not copy data
- Copy only specific keys from the data to the userData of notification message

The following table provides example values for the keysToPropagate option. In these examples, the **HotleadIdentifiedActionableEvent** data contains the keys **ORS Data, rule, strategy, some data, key_N1, key_X**.

Value of keysToPropagate	Keys which will be injected into notification message
all	All keys are copied: ORS Data, rule, strategy, some data, key_N1, key_X.

Value of keysToPropagate	Keys which will be injected into notification message
*	All keys are copied: ORS Data, rule, strategy, some data, key_N1, key_X.
key_*	The key_N1, key_X keys are copied.
no	No keys are copied.
rule, strategy	The rule, strategy keys are copied.
blank or empty	If the value of keysToPropagate 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 data of the HotleadIdentifiedActionableEvent.

Accessing Pacing Information from the Engagement Logic Strategy

In release 8.5, Web Engagement provides the Engagement Logic strategy with pacing data for the chat channel. 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 the **webengagement** open media interaction User Data by the Web Engagement 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, such as the data provided in business events) in the User Data of the **webengagement** interaction, as described above in the section on Accessing User Data from the webengagement Interaction and Passing it into Sub-flows.

Understanding How the Pacing Algorithm Works

A dedicated pacing algorithm serves each particular group of agents, so if you have 2 chat-oriented groups of agents, there will be 2 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
- 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 **pacing_chatCapacity** field.

For example:

```
pacing_chatCapacity:12
…
```

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 traffic that should be 'left' in
the system>
          },
          ...
       ],
       capacity: <count of allowed interactions for this channel>
    },
    ...
  ]
}
```

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

```
var pacingData = JSON.parse(_genesys.ixn.interactions[system.InteractionID].udata.pacing);
var currentChannel = undefined;
var channel = undefined;
var chatChannel = undefined;

for (channel in pacingData.channels) {
   currentChannel = pacingData.channels[channel];
   if (currentChannel.name=='chat') {
      chatChannel = currentChannel;
      break;
   }
}

var englishChatGroupCapacity = undefined;
var group = undefined;
var currentGroup = undefined;

if (chatChannel != undefined) {
    for (group in chatChannel.groups) {
       currentGroup = chatChannel.groups[group];
       if (currentGroup.name=='English Skill Group') {
         englishChatGroupCapacity = currentGroup.capacity;
         break;
       }
    }
}
```

Example of Using Pacing Information

Agents

Consider the following scenario where there are two chat 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)

The following group configuration options are set on the Web Engagement Cluster application:

- chatGroups = English Chat Group;Dutch Chat 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

- HitRate: 0.2
- Pending engagement invites: 0
- Reactive traffic is turned off

In this case, the results might look like this:

```
...
chatChannelCapacity : 5,
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
    }
  ]
}
```

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) from the pacing data.

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

- **Merijn**

    We can extract the capacity for the "Dutch Language Chat Group" (0) from the pacing data.

    In the decision workflow, it is not possible to engage Merijn on the chat channel.