Developer's Guide

Genesys Web Engagement 8.5.1

1/12/2022

Table of Contents

Genesys Web Engagement Developer's Guide

Welcome to the *Genesys Web Engagement 8.5.1 Developer's Guide*. This document provides information about how you can customize GWE for your website. See the summary of chapters below.

Architecture

Find information about Web Engagement architecture and functions.

High-Level Architecture

Engagement

Notification

Monitoring

Developing a GWE Application

Find procedures to develop an application.

Creating an Application

Instrumentation Script

Starting the Web Engagement Servers

Creating a Rules Package

Customizing a GWE Application

Find procedures to customize an application.

Customizing an Application

Creating Business Information

Customizing the Engagement Strategy

Customizing the Chat Routing Strategy

GWE Sample Applications

Learn about the Genesys Web Engagement Playground application.

Playground Application

Developer Tools

Find information about the GWE developer tools.

Integration

Learn how to integrate GWE with other components and media.

Simple ZAP Proxy

Advanced ZAP Proxy

Note: GWE also includes InTools, an application that helps you create, validate, and test DSL. You can read more about it in the User's Guide.

Integration with Second-Party and Third-Party Media

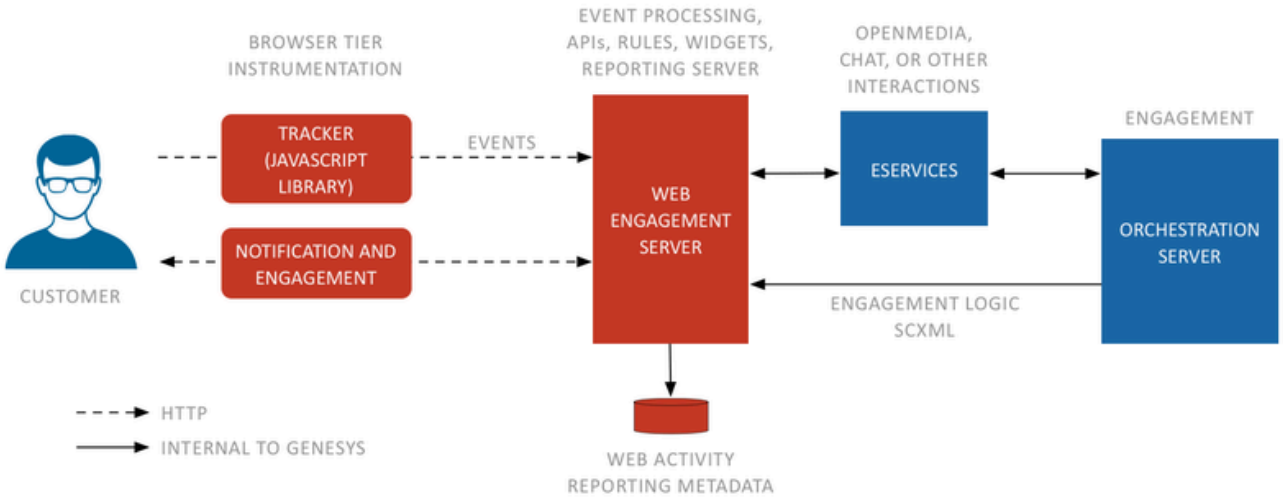
Serving Reactive Requests with Pacing Information

High-Level Architecture

Introduction

This article discusses the components that make up Genesys Web Engagement. Before you dive in, take a look at What is Web Engagement?

As mentioned in that article, Web Engagement has the following basic architecture:



As shown here, Web Engagement provides web services that connect your website with the Genesys contact center solution using:

- **Browser Tier Agents** (JavaScript code snippets) which are inserted into your web pages; they run in the visitor's browser and track their browsing activity.
- A **Web Engagement Server**, which includes the Web Monitoring Service and the Web Notification Service. This server is responsible for managing the data and event flow, based on a set of configurable rules and the visit's defined business events. It also stores data, submits information to the Genesys solution, and manages engagement requests to the Genesys contact center solution.

Browser Tier Agents

The Browser Tier Agents are implemented as JavaScript components that run in the visitor's browser. To enable monitoring on a web page, you create a short standardized section of JavaScript code, as specified in Configuring the Instrumentation Script, and then add this code snippet to the pages of

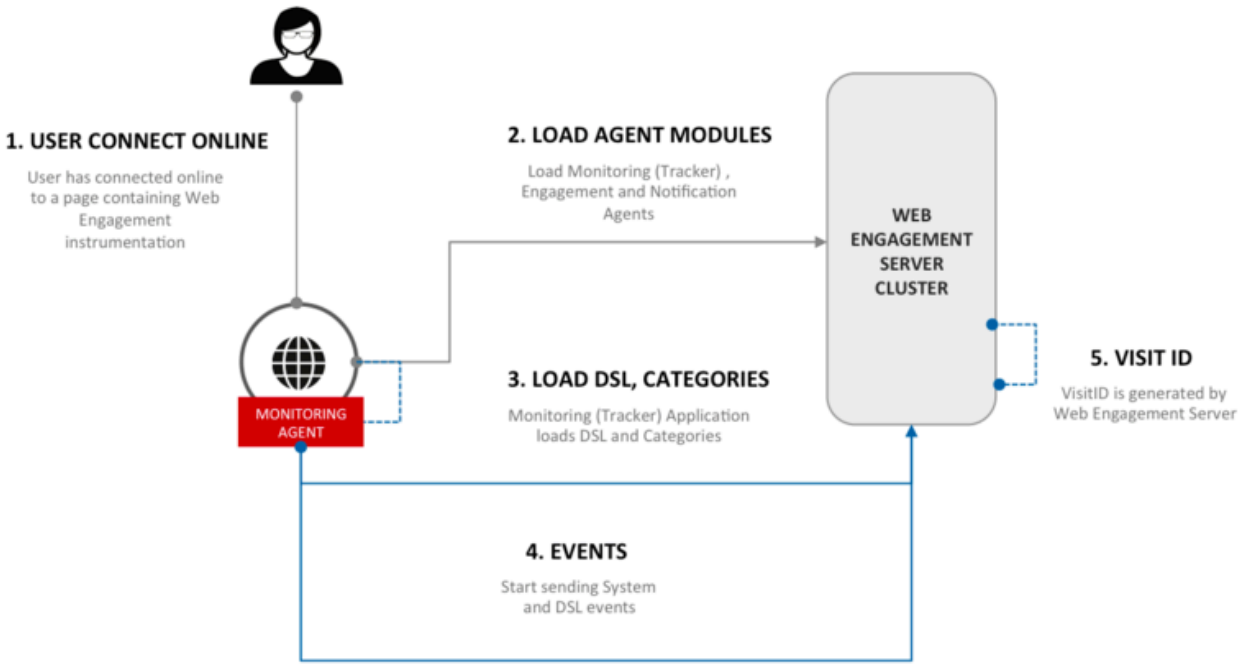
your site.

When a customer visits the webpage, the code retrieved within the page loads all the necessary artifacts like the JavaScript libraries and Domain Specific Language (DSL) that contains the definitions of your Business Events.

The DSL covers:

- The HTML elements to monitor.
- The custom business events to send to the Web Engagement Server.
- The data to include in the events.

The Browser Tier generates categorized standard System and custom Business events, defined in the DSL definitions, and sends them to the Web Engagement Server over HTTP.



Genesys Web Engagement provides the following browser tier agents:

- The **Monitoring Agent** records the web browsing activity. It generates basic system events such as VisitStarted, PageEntered, and additional custom business events, such as 'add-to-shopping cart'. These events are sent to the Web Engagement Server for further processing. For further information about events, see Event Workflow. For details about implementing monitoring, see Monitoring.
- The **Notification Agent** allows a web server to push data to a browser, without the browser explicitly requesting it, providing an asynchronous messaging channel between server and browser. It is used for presenting the engagement invite. For details about implementing notification, see Notification.

- The **Engagement Agent** provides the engagement mechanism and chat communication. For details about implementing engagement, see Engagement.

If you are only interested in Web Engagement's monitoring features, you need to configure your instrumentation script accordingly. See Configuring the Instrumentation Script for details.

Web Engagement Server

Working with the Browser Tier

The Genesys Web Engagement Server receives System and Business events from the browser's Monitoring Agent through its RESTful interface.

- **System** events track basic customer activities on your website. There are six of them, coming in two different flavors:
  - The **Visit-related** events, which are **VisitStarted**, **PageEntered**, and **PageExited**;
  - The **Identity-related** events, which are **SignIn**, **SignOut**, and **UserInfo**. See Visitor Identification for further details.
- **Business** events are additional custom events that you can create by implementing Advanced Engagement:
  - You create and define these events in the DSL loaded by the monitoring agents in the browser, using the Business Events DSL. For details about how to implement them, refer to Managing Business Events.
  - You can submit these events from your web pages by using the Monitoring Javascript API.

For details about how Business and System events are structured, see Events Structure.

The Monitoring JavaScript Agent gets a list of categories from the Web Engagement Server and categorizes each event, based on the event data, prior to sending it to the server. The integrated Complex Event Processing (CEP) engine processes incoming events against the business rules and creates actionable events when the required conditions are met. For more information on rules, consult the documentation for Genesys Rules System.

The Web Engagement Server also sends invitation notifications to the Notification Agent injected into the visitor's browser.

Hosting Static Resources

The Web Engagement Server is also responsible for hosting static resources, which are used in web applications such as Invite Widget, Chat Widget, and so on. Just as it does in version 8.5, Web Engagement 8.5.1 has two types of resources: modifiable and non-modifiable. However, Web Engagement 8.5.1 handles different types of resources differently than the 8.5 version.

Non-modifiable static resources

Non-modifiable resources should not be changed by customers and always stay inside of the Web

Engagement Server. They are available under the base path `http://{gwe server}/server/resources`

Non-modifiable resources include the Tracker JS Application, built-in version of Genesys CX Widgets, and Chat JS Applications. Here is part of the resources structure inside of the Web Engagement Server:

```
-resources
    -css
        -widgets
            widgets.min.css
            widgets.min.css.gz
            ...
    -js
        -build
            GPE.min.js
            GPE.min.js.gz
            GT.min.js
            GT.min.js.gz
            ...
        -widgets
            cxbus.min.js
            widgets.min.js
            widgets.min.js.gz
            ...
        chatAPI-noDeps.min.js
        chatAPI-noDeps.min.js.gz
        ...
```

Examples:

Tracker JS application can be accessed with the following request: `http://{gwe server}/server/resources/js/build/GPE.min.js` Legacy Chat JS API can be accessed with the following request: `http://{gwe server}/server/resources/js/chatAPI-noDeps.min.js` Built-in CX Widgets can be accessed with the following request: `http://{gwe server}/server/resources/js/widgets/widgets.min.js`

Modifiable static resources

These resources are all available to the newly created Web Engagement application in the ***GWE_installation*\apps*application_name*\resources** folder. Customers can modify these resources and also add new resources to the structure. After deploying an active application into Web Engagement Server, these resources will be available under the base path `http://{gwe server}/server/api/resources/v1`

Note: When a new GWE application is deployed, all resources belonging to previously deployed applications are removed. **Note:** During the deployment process, modifiable resources will persist in the DB layer and are synchronized between all GWE nodes.

Here is sample of the modifiable resources structure created by Web Engagement Server for the newly created application:

```
-resources
    -_composer_projects      // GRDT and SCXML Composer projects
    -drl                     // Rule files specific for this application (for example,
playground application comes with "ready-to-use" DRL file and does not require GRAT
deployment procedfure)
    -dsl
```

```
    domain-model.xml     // default DSL file
ads.html                 // sample advertisement widget
chatTemplates.html       // scripts for template-based modification of chat widget
chatWidget.html          // default chat widget
invite.html              // default invitation widget
```

You can add your own static resources under the Web Engagement Server, but Genesys recommends you do this only if the resources are related to the Genesys Web Engagement solution. Alternatively, you can host your static resources under a third-party server, as long as it supports all the features required for the Web Engagement solution.

Examples:

Default DSL file can be accessed with this request: `http://{gwe server}/server/api/resources/v1/dsl/domain-model.xml` Default advertisement widget can be accessed with this request: `http://{gwe server}/server/api/resources/v1/ads.html`

JSONP

The Web Engagement Server supports the JSONP protocol for all resources. JSONP stands for “JSON with Padding” and it is a workaround for loading data from different domains. It loads the script into the head of the DOM and thus you can access the information as if it were loaded on your own domain, by-passing the cross domain issue.

> **Tip**
> For more information about JSONP, see http://en.wikipedia.org/wiki/JSONP.

For example, for this request:

`http://{gwe server}/server/api/resources/v1/invite.html?obj=myObj&callback=myMethod`

the server returns following response body:

```
myObj.myMethod('<content of http://{gwe server}/server/api/resources/v1/invite.html>');
```

Cross-origin resource sharing

Cross-origin resource sharing (CORS) is a mechanism that allows many resources (for example, fonts, JavaScript, and so on) on a web page to be requested from another domain outside the domain from which the resource originated. In particular, JavaScript's AJAX calls can use the XMLHttpRequest mechanism. These "cross-domain" requests would otherwise be forbidden by web browsers due to the same-origin security policy.

> **Tip**
> For more information about cross-origin sharing, see http://en.wikipedia.org/wiki/

Cross-origin_resource_sharing.

GZIP

The Web Engagement Server can serve pre-compressed static content as a transport encoding and avoid the expense of on-the-fly compression. So if a request for **GPE.js** is received and the file **GPE.js.gz** exists, then it is served as **GPE.js** with a gzip transport encoding. By default, the Web Engagement solution ships all JavaScript resources in minified and pre-compressed version.

> **Tip**
>
> For more information about GZIP, see
>
> https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer#text-compression-with-gzip and
>
> http://en.wikipedia.org/wiki/HTTP_compression.

Working with the Enterprise Tier

The Web Engagement Server is also the engagement's entry point to the Genesys servers. It delivers web and visitor information to the contact center, which allows that information to be correlated with contact information.

On this end, the Web Engagement Server stores events, manages contexts and histories in its Cassandra database, and submits the appropriate data to the other Genesys servers.

When the Web Engagement Server is notified that it should present a proactive offer, it retrieves the engagement information, based on the visit attributes. Then, if the SCXML strategies allow it, the proactive offer is displayed.

If the visitor accepts, the Engagement service connects to the Genesys servers. Once the connection is established, the service manages the engagement context information across the visit.

The Web Engagement Server is also responsible for accepting rules deployed by the Genesys Rules Authoring Tool (GRAT).

Database and Reporting

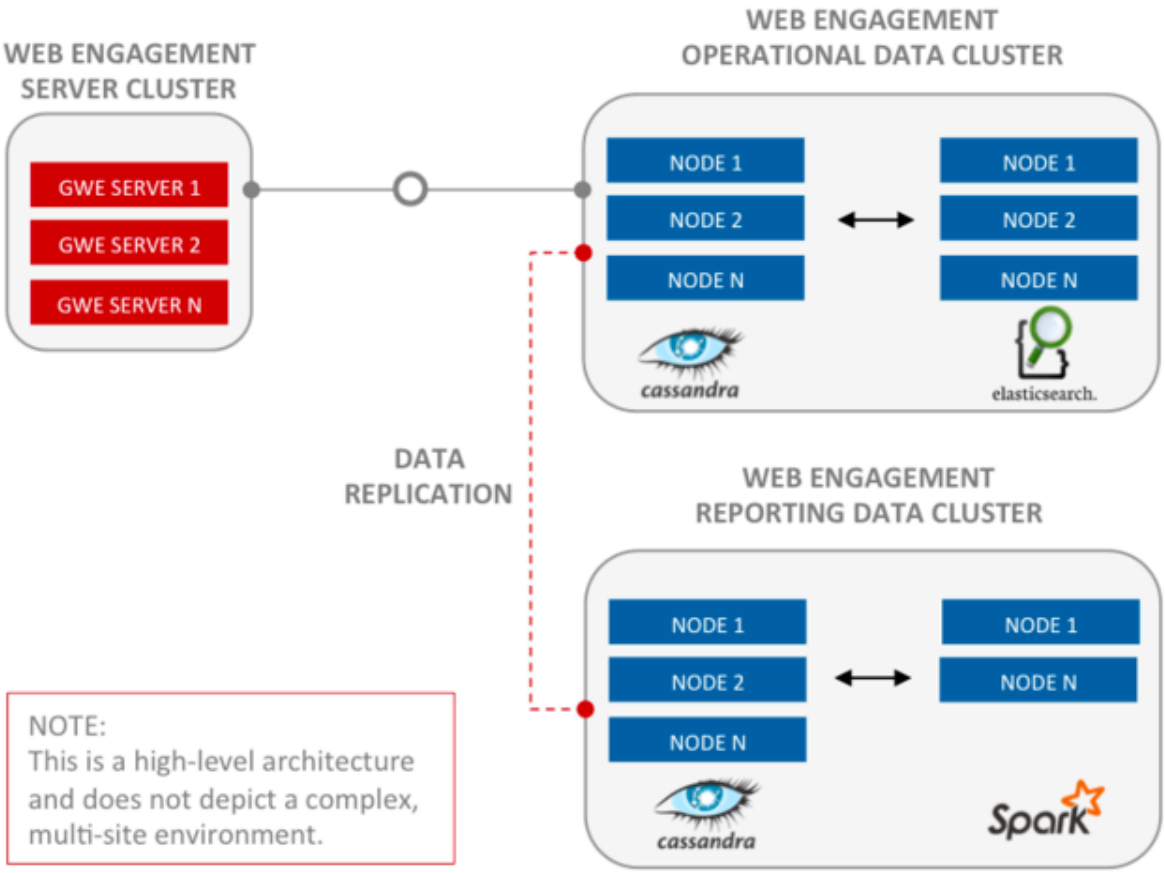
Web Engagement processes a large amount of data. To make this happen quickly enough, Genesys has combined three technologies into the database and reporting layers:

- Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

- Elasticsearch is a search server that provides a distributed, multitenant-capable full-text search engine with a RESTful web interface and schema-free JSON documents.
- Apache Spark is an open source cluster computing framework.

Cassandra and Elasticsearch clusters are used in the Operational Cluster that stores data for realtime processing. This Cassandra data is indexed by Elasticsearch for quick access, and the combined results are replicated in a separate Cassandra cluster in the Reporting Cluster. This Reporting Cluster uses a Spark cluster that massages the data in the Cassandra reporting cluster for more sophisticated reporting.

The following diagram provides a highly simplified view of how it all fits together.

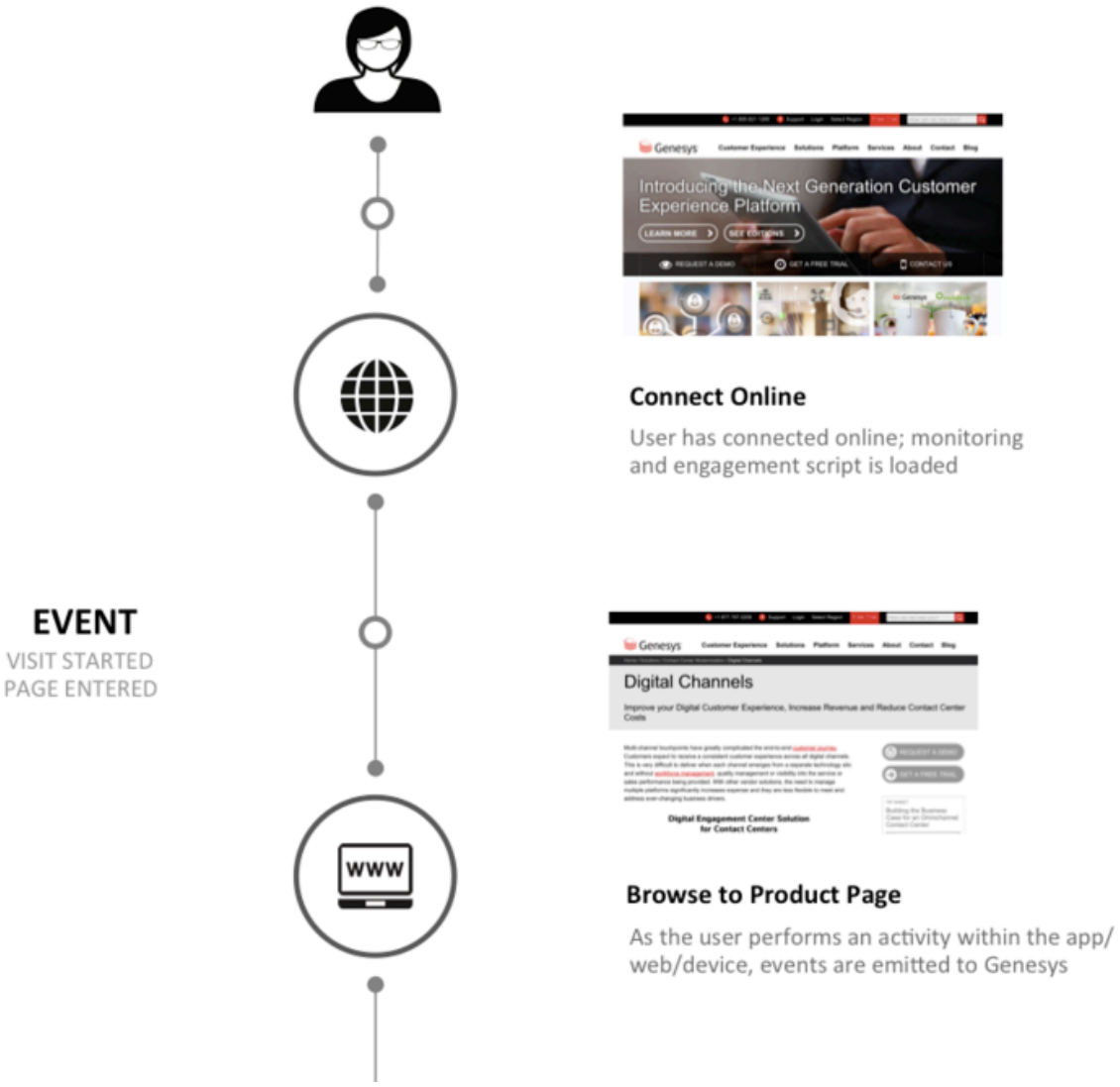


Event Workflow

The Genesys Web Engagement Server receives system and business events from the browser's Monitoring Agent. This event flow is used to create actionable events which generate requests to the

Genesys solution, and make the engagement, follow up, and additional actions with the Genesys solution possible. (Note that an actionable event does not always result in a notification—sometimes an action could be "do nothing.")

Here is a high-level view of this:

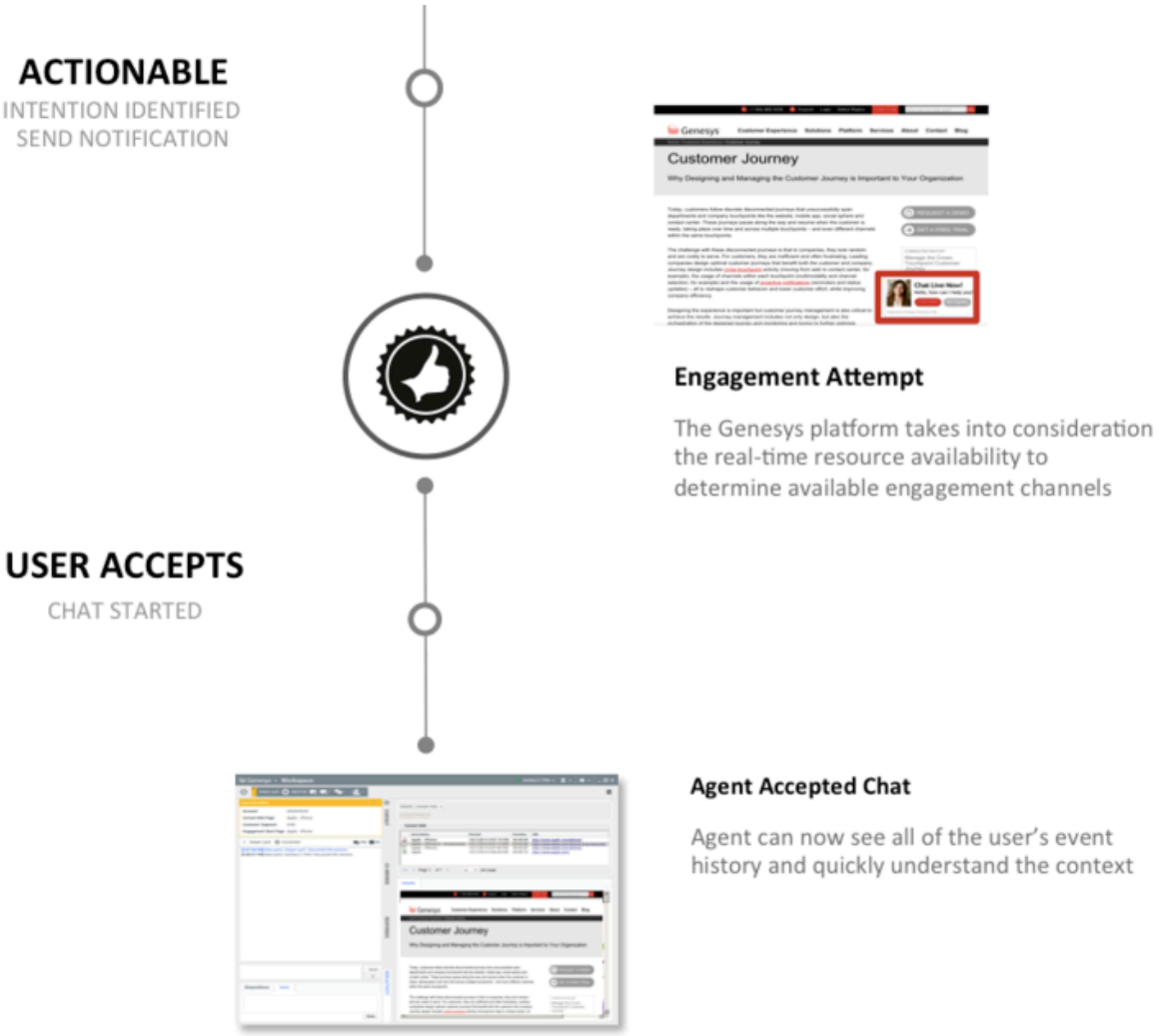


Requests a Demo by Submitting a Form

Events can be sequenced to determine customer intent and behaviors

Processing

Complex event processing rules are executed across the incoming events in real-time



As you can see, when a customer visits your website, he or she interacts with your web pages. The Monitoring Agent handles this traffic and translates it into the relevant System and Business events, according to your DSL and category information.

The agent then submits the events to the Web Engagement Server where the Complex Event Processing embedded in the server determines the actionable events ("Hot lead Identified" in the above figure) and carries out further processing. This includes the use of SCXML-based routing strategies to determine whether to proactively engage, to follow up, or to implement any other action.

Monitoring

The Monitoring Agent service records web browsing activity on your site. It generates basic system events such as VisitStarted, PageEntered, and additional custom business events, such as 'add-to-shopping cart'. Then it sends these events to the Web Engagement Server for further processing (you can read more about the structure of these events here).

To implement monitoring, you simply include the Monitoring Service JS script in your web pages. This short piece of regular JavaScript activates monitoring and notification functions by inserting one of the following scripts into the page: **GT.min.js, GTC.min.js, GPE.min.js**. The script depends on your requirements — see Configuring the Instrumentation Script for details. The JavaScript asynchronously loads the application into your pages, which means that Monitoring Service JS does not block other elements on your pages from loading.

Basic Configuration

The simplest way to get the Monitoring Service JS for your site is by creating and adding a script to your web pages. See Configuring the Instrumentation Script for details.

> **Important**
>
> If you plan to use Web Engagement chat, make sure to include the Chat JS Application script into your web pages, as well. See Engagement for details.

Advanced Configuration

Once you have a basic configuration script, you can use it "as is" or implement the advanced configuration options to customize the script to suit your requirements. See Configuring the Instrumentation Script for details.

Monitoring JS API

You can also take a highly customized approach and use the Monitoring JS API to submit events to the Web Engagement Server. You can submit UserInfo, SignIn, SignOut, and even your own custom business events using this API. For example, you can use the API to identify visitors on your website. See Visitor Identification for details.

Related Links

- Visitor Identification
- Events Structure

Visitor Identification

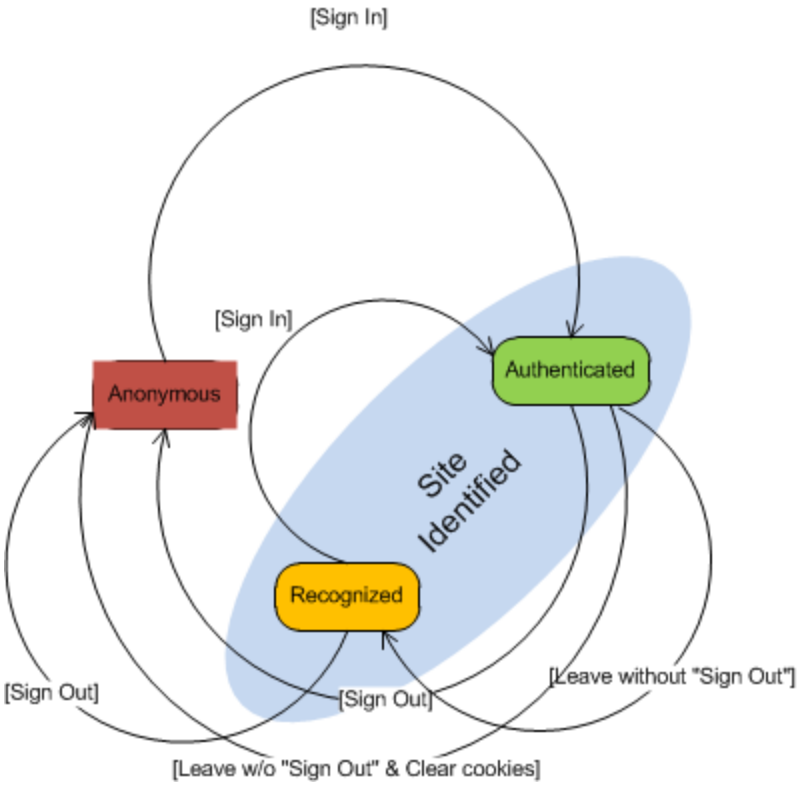
Overview

Genesys Web Engagement allows you to capture visitor activities on your website and to build a complete history of the visitor's interactions with your contact center.

When a visitor browses your website, the tracking code submits System events to the Web Engagement servers that constitute a visit, such as VisitStarted, PageEntered, SignIn, UserInfo, and so on. The association or relationship between the visit and the visitor is based on the flow derived from System events, in addition to the information retrieved from the Contact Server. In the end, you can access visit history through the Event Resource in the History REST API.

To associate the visitor with the visit, Genesys Web Engagement must "identify" the visitor as one of three possible states:

- **Authenticated** — The visitor logged in to the website with a username and password. The username can be an e-mail address, an account name or other similar identifier, depending on your website. When a user is authenticated, Genesys Web Engagement can maintain an association between the visitor and the visit.
- **Recognized** — The visitor closed the browser window and did not log out, but cookies are saved. The next time the visitor comes to the website, the website can submit cookie-based user information, which contains the **userId**.
- **Anonymous** — The visitor is anonymous.



Visitor states

Genesys Web Engagement relies on your website to trigger the transitions between visitor states. You can do this by updating the tracking code with the following events in the Monitoring JS API:

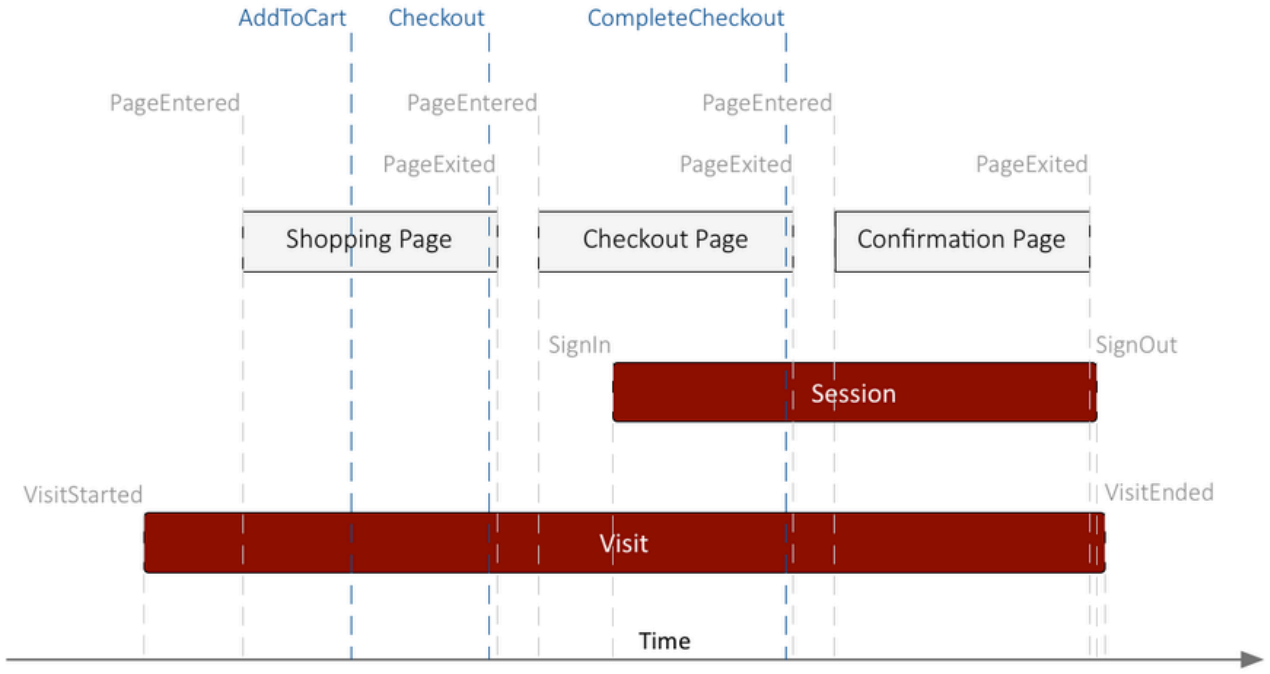
- _gt.push(['event', 'SignIn', { data: options }]) or _gt.push(['event', 'sendSignIn', options]) — Send this event when the user is authenticated by the website. This allows the system to identify the user and creates a new "session" with a **sessionId** that is unique to a visit and will last the duration of the visit. Only Authenticated visitors have an associated **sessionId**.

- _gt.push(['event', 'SignOut', { data: options }]) or _gt.push(['event', 'sendSignOut', options]) — Send this event when the user logs out of the website.
  Note: The **sessionId** lasts for the duration of the authenticated user's visit to your website. It is stored in a cookie and sent with every event that occurs between SignIn and SignOut, and is changed automatically after every SignIn event.

- _gt.push(['event', 'UserInfo', { data: options }]) or _gt.push(['event', 'sendUserInfo', options]) — Send this event when the user visits your website after closing the browser window on an authenticated session. For details, see Recognized Visitors.

Visitor Event Timeline

The figure below shows the timeline for events that take place when a visitor browses your website.



All visitors to your website are identified with a **visitId**, which can be used to associate the visitor to events, such as PageEntered or PageExited, during the span of the visit.

Accessing Visitor Information

The History REST API is a RESTful interface for accessing visit and identity information — in the form of a collection of JSON objects — via POST and GET HTTP requests:

- The visit resource represents the sequence of pages that a given visitor went through.
- The identity resource contains information about authenticated and recognized visitors.
- The page resource contains information about browsed pages. If a visitor revisits a page, a new page resource is created.
- The event resource contains information about System and Business events. For details about how these events are structured, see Events Structure.

Authenticated Visitors

When the visitor is Authenticated on the website, you should use the_gt.push(['event', 'SignIn', { data: options }]) or _gt.push(['event', 'sendSignIn', options]) event so that Genesys Web Engagement can start a new session. When the Web Engagement Server receives the related command, it creates a new session for the current visit. This process is completely transparent to the customer. The identifying information used to log in (for instance, the email address) is available in the SignIn event and is used to:

- Create the **identityId** or search the visitor's identity resource.
- Associate the visitor with a contact in the Genesys solution.

Recognized Visitors

When an Authenticated visitor closes the browser window without signing out and then later revisits your site, you can use the _gt.push(['event', 'UserInfo', { data: options }]) or _gt.push(['event', 'sendUserInfo', options]) command to tell Genesys Web Engagement that the visitor is now Recognized.

You will need to send the **userId** in the **_gt.push(['event', 'UserInfo', { data: options }])** or **_gt.push(['event', 'sendUserInfo', options])** event. How you track the **userId** depends on your website. For example, you could create a persistent cookie to store the **userId** when a visitor logs in to you website. Then when a visitor first browses your site, you could check the cookie and call the **_gt.push(['event', 'UserInfo', { data: options }])** or **_gt.push(['event', 'sendUserInfo', options])** event if the cookie contains the **userId**. There are many possible scenarios - the best implementation is entirely dependent on your website and its workflow.

> ### Important
> The visitor's identity cannot be guaranteed in the Recognized state. For instance, another member of the visitor's family could be browsing the website with the same computer.

Anonymous Visitors

If the visitor is not Authenticated or Recognized, he or she is treated as Anonymous. The visitor's activity on the website—including events and pages visited—is still associated with the visit.

Events Structure

Overview

When the Tracker Application monitors the current web page, it generates a series of events, which are represented in JSON format.

There are two available event types:

- SYSTEM — These events are generated automatically and cannot be configured.
- BUSINESS — These are additional custom events you can create.

> **Important**
> You can configure when an event should be generated by customizing the DSL, but if you need more flexibility you can use the Monitoring JS API.

Common Event Structure

The common event structure is a scaffold for generating System and Business events. In table below, **data** represents the common structure that is included in both event types.

Field	Type	Description
eventType	String	The event type: `BUSINESS` or `SYSTEM`.
eventName	String	The required event name.
eventID	String	The generated UUID that is used to identify the event.
pageID	String	The generated UUID that is used to identify the page.
timestamp	Number	The time stamp for when the event was generated. This is taken from the browser.
category	String	A list of categories separated by semicolons. For more information, see Managing Categories.
url	String	The URL of the page where the event was triggered.

Field	Type	Description
languageCode	String	The current language. This can be configured using the **languageCode** configuration option in the instrumentation script.
globalVisitID	String	globalVisitID is a anonymously identifier of a particular device or browser.
visitID	String	visitID represents a particular session in the browser.
data	Object	Container for additional data. Which depends on event type and name. See appropriate event below

Example of the Common Event Structure

```
{
  "eventName": "PageEntered",
  "eventID": "44D25DDB78174DEC8F33E28F96428336",
  "pageID": "9A1AD4389AC34F0A86D3EB04E50D6137",
  "timestamp": 1413979605190,
  "category": "my-category",
  "url": "http://www.genesys.com/products",
  "languageCode": "en-US",
  "globalVisitID": "b5a93936-b2a4-4042-a5e6-0a2b9681c1a9",
  "visitID": "8bd4bbb5-3b1d-4647-9ede-37820b88e343",
  "data": {
    ...
  }
}
```

System Event Structure

System events have specific values for the following fields:

Field	Type	Description
eventType	String	`SYSTEM`
eventName	String	The following values are possible: `PageEntered`—generated when the user enters a page `PageExited`—generated when the user changes location or closes a page `SignIn`—generated when the user signs in

Field	Type	Description
		`SignOut`—generated when the user signs out `UserInfo`—generated when the user signs in `VisitStarted`—generated when the visit is identified
data	Object	This field should contain specific information, described in System data below.

System data

The value of the System event's **data** field can vary depending on the name of System event. The following sections provide details about the **data** provided for each event name.

VisitStarted

The `VisitStarted` event expands the System event structure with the following value for **data**:

data field	Type	Description
userAgent	String	The `window.navigator.userAgent` value. This contains information about the name, version, and platform of the browser.
screenResolution	String	The screen resolution at the moment the event is generated. The format is width x height. For example: "1440x900"
language	String	The language code from `window.navigator`, retrieved from the first available of the following objects: `window.navigator.language || window.navigator.browserLanguage || window.navigator.userLanguage || window.navigator.systemLanguage`
timezoneOffset	String	The timezone offset in milliseconds.
ipAddress	String	The client IP address.

```
{
        "eventType":"SYSTEM",
        "eventName":"VisitStarted",
        "eventID":"5E1BA21F69F149F280B028385DF16DC3",
        "pageID":"300E084345EC412F879D5A835F7CA4F6",
        "timestamp":1414074819648,
        "category":"my-category",
```

```
        "url":"http://www.genesys.com/products",
        "languageCode":"en-US",
        "globalVisitID":"301438c2-3139-4035-aac0-1c9c8a60c481",
        "visitID":"9ccd9489-6a94-4b45-8813-c1cca010d443",
        "data":{
                "userAgent":"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:32.0) Gecko/20100101
Firefox/32.0",
                "screenResolution":"1680x1050",
                "language":"en-US",
                "timezoneOffset":-10800000,
                "ipAddress":"123.45.67.890"
        }
}
```

PageEntered

The PageEntered event expands the System event structure with the following value for **data**:

data field	Type	Description
urlReferrer	String	A `window.document.referrer` value. The referrer property returns the URL of the document that loaded the current document.
localTime	String	The string representation of the time in the browser when the event was generated.
title	String	The page title, taken from `window.document.title`.

```
{
  "eventType": "SYSTEM",
  "eventName": "PageEntered",
  "eventID": "44D25DDB78174DEC8F33E28F96428336",
  "pageID": "9A1AD4389AC34F0A86D3EB04E50D6137",
  "timestamp": 1413979605190,
  "category": "my-category",
  "url": "http://www.genesys.com/products",
  "languageCode": "en-US",
  "globalVisitID": "b5a93936-b2a4-4042-a5e6-0a2b9681c1a9",
  "visitID": "8bd4bbb5-3b1d-4647-9ede-37820b88e343",
  "data": {
    "urlReferrer": "http://www.genesys.com",
    "localTime": "Wed Oct 22 2014 15:06:45 GMT+0300 (FLE Daylight Time)",
    "title": "English"
  }
}
```

PageExited

The PageExited event does not have additional data. The event structure is the same as the System event structure, but with the PageExited event name specified:

```
{
  "eventType": "SYSTEM",
  "eventName": "PageExited",
```

```
  "eventID": "E8E6F0926F3642BF889DA5ED4342EFA7",
  "pageID": "9A1AD4389AC34F0A86D3EB04E50D6137",
  "timestamp": 1413982730013,
  "category": "my-category",
  "url": "http://www.genesys.com/products",
  "languageCode": "en-US",
  "globalVisitID": "b5a93936-b2a4-4042-a5e6-0a2b9681c1a9",
  "visitID": "8bd4bbb5-3b1d-4647-9ede-37820b88e343",
  "data": {}
}
```

UserInfo

The `UserInfo` event expands the System event structure with the following value for **data**:

data field	Type	Description
userID	String	A unique persistent string identifier that represents a user or signed-in account across devices.

```
{
   "eventType":"SYSTEM",
   "eventName":"UserInfo",
   "eventID":"532BC42B99C341578639A1DF1F2A45D9",
   "pageID":"C90206CA44A2401F9408A1581EF0E258",
   "timestamp":1419437657401,
   "category":"",
   "url":"http://www.genesys.com/products",
   "languageCode":"en-US",
   "globalVisitID":"c9b891e4-ae04-493b-b554-4eba19ad7c58",
   "visitID":"b5c87b28-a00e-4461-961b-d6a01b754838",
   "data":{
      "userID":"user@genesyslab.com",
      "name":"Bob",
      "sex":"male",
      "age":30
   }
}
```

SignIn

The `SignIn` event expands the System event structure with the following value for **data**:

data field	Type	Description
userID	String	A unique persistent string identifier that represents a user or signed-in account across devices.

```
{
   "eventType":"SYSTEM",
   "eventName":"SignIn",
   "eventID":"DE6826972BDF4820B03FAF5BB7945426",
   "pageID":"C90206CA44A2401F9408A1581EF0E258",
   "timestamp":1419437874950,
   "category":"",
   "url":"http://www.genesys.com/products",
```

```
   "languageCode":"en-US",
   "globalVisitID":"c9b891e4-ae04-493b-b554-4eba19ad7c58",
   "visitID":"b5c87b28-a00e-4461-961b-d6a01b754838",
   "data":{
      "userID":"user@genesyslab.com",
      "name":"Bob",
      "sex":"male",
      "age":30
   }
}
```

SignOut

The `SignOut` event does not have additional data. The event structure is the same as the System event structure, but with the `SignOut` event name specified:

```
{
   "eventType":"SYSTEM",
   "eventName":"SignOut",
   "eventID":"3CE3204E697640A7986C70CA97F0945C",
   "pageID":"C90206CA44A2401F9408A1581EF0E258",
   "timestamp":1419437925162,
   "category":"",
   "url":"http://www.genesys.com/products",
   "languageCode":"en-US",
   "globalVisitID":"c9b891e4-ae04-493b-b554-4eba19ad7c58",
   "visitID":"b5c87b28-a00e-4461-961b-d6a01b754838",
   "data":{
   }
}
```

Business Event Structure

Business events have the same structure as the common event structure, with additional data specified in the DSL configuration. For example, if your DSL (**domain-model.xml**) has the following event generation rules:

```
<event id="TimeoutEvent10" name="Timeout-10" condition="" postcondition="document.hasFocus()
=== true">
      <trigger name="TimeoutTrigger" element="" action="timer:10000" type="timeout" url=""
count="1" />
      <val name="myValueName" value="'myValue'"></val>
</event>
```

Then the generated Business event is expanded with the additional data:

```
{
  "eventType": "BUSINESS",
  "eventName": "Timeout-10",
  "eventID": "11030C008B3D45ACADFB32A1B4E01122",
  "pageID": "B501B6EE57EF4E2AA05379D468E772D6",
  "timestamp": 1413990905565,
  "category": "",
  "url": "http://www.genesys.com/products",
  "languageCode": "en-US",
  "globalVisitID": "b5a93936-b2a4-4042-a5e6-0a2b9681c1a9",
  "visitID": "8bd4bbb5-3b1d-4647-9ede-37820b88e343",
```

```
  "data": {
    "myValueName": "myValue"
  }
}
```

Notification

The Notification Agent provides the browser with the asynchronous notification of the engagement offer by opening an engagement invite. It opens the engagement window.

To implement notification, simply include the Notification Service JS script in your web pages. This short piece of regular JavaScript activates monitoring and notification functions by inserting one of the following scripts into the page: **GT.min.js, GTC.min.js, GPE.min.js**. The script depends on your requirements — see Configuring the Instrumentation Script for details. The JavaScript asynchronously loads the application into your pages, which means that Notification Service JS does not block other elements on your pages from loading.

Basic Configuration

The simplest way to get the Notification Service JS for your site is by creating and adding a script to your web pages. See Configuring the Instrumentation Script for details.

> **Important**
> If you plan to use Web Engagement chat, make sure to include the Chat JS Application script into your web pages, as well. See Engagement for details.

Advanced Configuration

Once you have the basic configuration script, you can use it as is or implement the advanced configuration options to configure the script to suit your requirements. See Configuring the Instrumentation Script for details.

Notification Service REST API

You can use the Notification Service REST API to reach your entire user base quickly and effectively with notifications that are delivered to your web pages. For details, see Notification Service REST API in the API Reference.

Engagement

The Engagement Agent provides the engagement mechanism — proactive/reactive chat communication.

Chat JS Application

To implement chat, you simply include the Chat JS Application script in your web pages. This short piece of regular JavaScript activates chat functions by inserting the **GWC.min.js** script into the page. The JavaScript asynchronously loads the application into your pages, which means that Chat JS Application does not block other elements on your pages from loading.

Basic Configuration

The simplest way to get the Chat JS Application for your site is by creating and adding a script to your web pages. See Configuring the Instrumentation Script for details.

Advanced Configuration

The Chat JS Application script consists of two parts: **script loader** and **configuration**. The script loader part actually loads the **GWC.min.js** script, while the configuration part sets options that control things like window size and localization.

Script Loader

To load Chat JS Application, you just need to include a short piece of regular JavaScript, the script loader, in your HTML. That JavaScript will asynchronously load the application into your pages, which means that Chat JS Application will not block other elements on your web page from loading.

For example, your script loader code might look like this:

```
//Script loader
(function(v) {
    if (document.getElementById(v)) return;
    var s = document.createElement('script'); s.id = v;
    s.src = (   'https:' == document.location.protocol ? 'https://<Web Engagement Server
host>:<Web Engagement Server secure port>' :
                'http://<Web Engagement Server host>:<Web Engagement Server port>') +
'/server/resources/js/build/GWC.min.js';
    s.setAttribute('data-gwc-var', v);
    (document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_gwc');
```

> **Important**
>
> The above example uses _gwc as the configuration global variable — see the "Configuration" section below for details.

Configuration

By default the chat application uses the _gwc global variable (you can change this in the script loader) that is created before Chat JS Application script loader is actually added to the page. Some of the options you set in the configuration code can be overwritten in the Chat Widget JS API methods (startChat(options) and restoreChat(options) for a particular chat session, if the parameter name matches the option name.

For example, your configuration code might look like this:

```
/* Configuration */
var _gwc = {widgetUrl: 'http://<Web Engagement Server host>:<Web Engagement Server
port>/server/api/resources/v1/chatWidget.html',
            serverUrl: 'http://<Web Engagement Server host>:<Web Engagement Server
port>/server/cometd'};
```

Options

Option	Type	Default Value	Mandatory	Description
serverUrl	string	undefined	yes, when default "transport" is used	URL of the CometD chat server for default (built-in) CometD transport.
widgetUrl	string	undefined	yes, when "embedded" is set to false ("popup" mode)	URL of the chat widget HTML that is open in an external window when operating in "popup" mode. By default, the chat widget is stored under the Web Engagement Server and is available at the following URL: http://{gwe_server}:{server_port}/server/api/resources/v1/chatWidget.html; however, you can store the chatWidget.html file as a static resourced under any third-party server.
embedded	boolean	false	no	Sets chat mode of operation: "embedded" (chat widget is rendered directly on a page) or "popup" (chat opens in a separate browser window). Default is "popup". Pass the value true to switch to "embedded" mode.
localization	object or string or function	undefined	no	Provider for custom localization, which will be one of the following: • A JavaScript object containing localization data

Option	Type	Default Value	Mandatory	Description
				• A function that returns an object containing localization data • A function that accepts a callback and calls it with an object containing localization data • The URL for an external JSON file containing localization data If omitted, the default English localization will be used. See Localization for more on how to localize the chat widget.
windowSize	object {width: <number>, height: <number>}	{ width: 400, height: 500 }	no	Size of external chat window when operating in "popup" mode.
windowName	string	genesysChatWindow	no	A string name for the new window that will be passed to the `window.open` call when opening chat widget window. For details, see https://developer.mozilla.org/en-US/docs/Web/API/Window.open. **Note:** If you need to support Internet Explorer versions 8 or 9, **windowName** must not contain either hyphens ("-") or spaces (" "), as documented at http://stackoverflow.com/questions/710756/ie8-var-w-

Option	Type	Default Value	Mandatory	Description
				window-open-message-invalid-argument.
windowOptions	object	The value of the windowSize option.	no	An object containing window options that are passed to the `window.open` call when opening chat widget window. You can pass any window options, such as position (`top, left`), whether to show browser buttons (`toolbar`), location bar (`location`), and so on. For details about possible window options, see (https://developer.mozilla.org/en-US/docs/Web/API/Window.open. All options are converted to a string that is passed to the `window.open` call.
debug	boolean	false	no	Set to true to enable chat debugging logs (by default standard console.log is be used, see the "logger" option if you want to override that).
logger	function	console.log	no	Pass a function that will be used for chat logging (if debug is set to true) instead of the default console.log. The function has to support the interface of the console.log — it must accept an arbitrary number of arguments and argument types. To use the custom logging function in a

Option	Type	Default Value	Mandatory	Description
				separate window, you have to pass it directly on the widget page to the startChatInThisWindow method. Important The "logger" function works only for the Chat Widget JS API context.
registration	boolean or function	true	no	By default chat starts with a built-in registration form (that you can customize using ui.onBeforeRegistration). Pass the value false to disable this default built-in registration form. See Custom registration in the Chat Widget JS API for details.
userData	object	undefined	no	Can be used to directly attach necessary UserData to a chat session.
createContact	boolean	true	no	Determines whether new contact should be created from registration data if it doesn't match any existing contact. Only effective if registration data is present (collected either by built-in or custom registration workflow). See createContact in the Chat Widget JS API for details.

Option	Type	Default Value	Mandatory	Description
maxOfflineDuration	number	5	no	Time (in seconds) during which state cookies are stored after page reload/ navigation. If cookies expire, the chat is not restored. Basically, this option means "how long shall the chat session live after the user leaves my website?"
ui	boolean or object	true	no	Pass the value false to disable the chat widget UI completely. Or pass an object with "hook" functions that can modify the built-in UI. See ui in the Chat Widget JS API for details.
transport	object	undefined	no	Custom transport instance (for example, REST-based).
disableWebSockets	boolean	false	no	By default, chat attempts to use WebSockets to connect to the server. When the WebSocket connection is unavailable (for example, if your load balancer doesn't support WebSockets), chat switches to other, HTTP-based, means of communication. This might take some time (a matter of seconds, usually), so if you want to speed up the process, you can disable WebSockets for chat by

Option	Type	Default Value	Mandatory	Description
				passing true to this option. Important This option is only effective with default (built-in) transport.
templates	string	undefined	no	The URL of the HTML files containing templates that are used to render the chat widget. The request is made via either JSONP or AJAX, following the same logic as for localization files (see Localization in the Chat Widget JS API). Default templates are included into the JavaScript source, so by default no requests are made to load them. The template system is based on the popular lodash / underscore templates: http://lodash.com/docs#template, http://underscorejs.org/#template
autoRestore	boolean	true	no	On every page reload/ navigation, chat automatically attempts to restore the chat widget using the restoreChat method in the Chat Widget JS API. You can use this option to disable this behavior if you want more control over chat widget restoration.

Option	Type	Default Value	Mandatory	Description

onReady	array or function	undefined	no	This field is a callback function fired when the application has initialized. The Chat Widget JS API object is provided as the first argument of the callback function.
				
				`_gwc.onReady.push(function(chatAPI) {`
				`  alert('Chat application ready!');`
				`});`
				
				If you use `_gwc.onReady.push`, make sure that `onReady` is defined as an array.
				
				`var _gwc = {`
				`  ...`
				`  onReady: []`
				`};`

Configuration Examples

Basic configuration for proactive engagement integration

```
/* Configuration */
var _gwc = {widgetUrl: 'http://<Web Engagement Server host>:<Web Engagement Server
port>/server/api/resources/v1/chatWidget.html'};

// Script loader
(function(v) {
    if (document.getElementById(v)) return;
    var s = document.createElement('script'); s.id = v;
    s.src = ('https:' == document.location.protocol ? 'https://<Web Engagement Server
host>:<Web Engagement Server secure port>':
        'http://<Web Engagement Server host>:<Web Engagement Server port>') + '/server/
resources/js/build/GWC.min.js';
    s.setAttribute('data-gwc-var', v);
    (document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_gwc');
```

Basic configuration for reactive chat

```
/* Configuration */
var _gwc = {widgetUrl: 'http://<Web Engagement Server host>:<Web Engagement Server
port>/server/api/resources/v1/chatWidget.html',
            serverUrl: 'http://<Web Engagement Server host>:<Web Engagement Server
port>/server/cometd'};

// Script loader
(function(v) {
    if (document.getElementById(v)) return;
    var s = document.createElement('script'); s.id = v;
    s.src = ('https:' == document.location.protocol ? 'https://<Web Engagement Server
host>:<Web Engagement Server secure port>':
        'http://<Web Engagement Server host>:<Web Engagement Server port>') + '/server/
resources/js/build/GWC.min.js';
    s.setAttribute('data-gwc-var', v);
    (document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_gwc');
```

Advanced configuration for chat application

```
/* Configuration */
var _gwc = {
    serverUrl: 'http://<Web Engagement Server host>:<Web Engagement Server port>/server/
cometd',
    widgetUrl: 'http://<Web Engagement Server host>:<Web Engagement Server port>/server/api/
resources/v1/chatWidget.html',
    autoRestore: true,
    debug: false,
    embedded: true,
    createContact: true,
    localization: 'http://<Web Engagement Server host>:<Web Engagement Server port>/server/
api/resources/v1/locale/chat-fr.json',
    windowSize: { width: 400, height: 500 },
        windowName: 'myWindowName',
        windowOptions: {
        left: 0,
        top: 0
```

```
    },
    /* Callbacks */
    onReady: [function (chatAPI) {
        var options = {
            registration: true
        };
        chatAPI.startChat(options);
    }]
};
// Script loader
(function(v) {
    if (document.getElementById(v)) return;
    var s = document.createElement('script'); s.id = v;
    s.src = ('https:' == document.location.protocol ? 'https://<Web Engagement Server
host>:<Web Engagement Server secure port>':
        'http://<Web Engagement Server host>:<Web Engagement Server port>') + '/server/
resources/js/build/GWC.min.js';
    s.setAttribute('data-gwc-var', v);
    (document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_gwc');
```

> **Tip**
>
> For more information about the start parameters, see the Chat Widget JS API

Chat JS Application API

The Chat JS Application API is provided by the Chat Widget JS API component. The API object provides two functions: startChat(options) and restoreChat(options). To access the API, use the onReady option in the Chat JS Application configuration.

Reactive Chat

The following example shows how you can start reactive chat on a button click using the `startChat` method.

```
$('#startChatButton1, #startChatButton2, #startChatButton3').click(function () {
    _gwc.onReady.push(function (chatAPI) {
        chatAPI.startChat();
    });
});
```

If you want to provide monitoring information to the chat session, you should attach the **visitID** and **pageID** from the Tracker Application to the chat interaction.

```
$('#startChatButton1, #startChatButton2, #startChatButton3').click(function () {
    _gwc.onReady.push(function (chatAPI) {
        _gt.push(['getIDs', function (IDs) {
            chatAPI.startChat({userData: {visitID: IDs.visitID, pageID: IDs.pageID}});
        }]);
    });
});
```

How To

Auto-generate an e-mail address based on the visitID

Use the Tracker JS Application and the Chat JS Application together:

```
_gwc.onReady.push(function (chatAPI) {
    _gt.push(['getIDs', function (IDs) {

        /* Start chat with generated email */
        chatAPI.startChat({ userData: {
            visitID: IDs.visitID,
            pageID: ID.pageID,
            email: IDs.globalVisitID + '@anonymous.com'
        }});
    }]);
});
```

Customization

For details about how to customize the callback widget, see Customizing the Browser Tier Widgets.

Application Development

Overview

Developing an application for Genesys Web Engagement is the process of defining all the components deployed through the Web Engagement Servers to implement Web Engagement features in your Genesys contact center, and to add Web Engagement to your website.

When you create and configure your application, you create all the materials that are used to generate the actionable events: customized business information, conditions, and engagement strategies. As a result of an actionable event, the Web Engagement servers engage the visitor with a chat invite or another widget, such as an advertisement. Your application also contains the widgets for managing these invites, including a registration form submitted to anonymous customers who accept the invitation.

The provided script tools create your application in the **apps** folder where Web Engagement is installed. Your newly created application includes all the default rule templates, logic (SCXML), and events (DSL), in addition to web-specific data and engagement widgets. You can customize the data and widgets, and then deploy your application so all changes take effect.

Your new application can be adapted to work with two different engagement models:

- Simple Engagement Model — This type of engagement model works with default Web Engagement capabilities, and provides customization through categories and rules.
- Advanced Engagement Model — This type of engagement model works with the same set of entities as the simple engagement model, but also uses customer-specific business events (that are defined in your DSL) and event-based capabilities to implement rules.

Application Development Workflow

The following diagram describes the development workflow for a Web Engagement application.

1. **Create your application**
   Tool: Web Engagement Scripts

   Description: For each application you must use script tools to create and configure your customized Web Engagement application.

2. **Start the Web Engagement Servers**
   Tool: Web Engagement Scripts.

   Description: To be able to deploy resources or rules and use the Web Engagement Management tool, you must start the Web Engagement Servers.

3. **Create Customized Business Information**
   Depending on the engagement model that you implement, you must define business information

specific to your web pages that will be used to submit actionable events and web contexts to the Genesys Solution.

- **Create categories** (Simple Model)
  Tool: Web Engagement Management interface

  Description: The categories contain business-related information to link your application with your web pages. They are used as parameters to set up conditions on events and generate actionable items. You can modify category information at run-time. The monitoring agent requests a list of categories from the Web Engagement Server every time a new web page is loaded or reloaded. If categories have not changed since the previous request, Web Engagement Server returns an HTTP response 304 "Not Modified" and the categories are reloaded from the browser cache.

- **Create Business Events.** (Advanced Model)
  Tool: Text editor / Chromium InTools

  Description: You can create your own business events as lists of DSL items, which are loaded by the monitoring agent. Then, these events are sent to the Web Engagement Server and processed in the same manner as regular system events. To apply the DSL changes, you need to redeploy the application with the modified DSL into the Web Engagement Server. You can also test the changes at run-time with Chromium InTools.

4. **Publish Rules Template and Customize Logic**
   Tool: Genesys Rules Development Tool / Composer

   Description: You must publish a Web Engagement rules template before you can create rules. If you want to, you can also customize your logic by Customizing the SCXML Strategies, and you can also customize both the Browser Tier Widgets and the Chat Routing Strategy.

5. **Deploy your application**
   Tool: Web Engagement Scripts

   Description: If you create a new application or modify the SCXML, the DSL, or the logic of your application, you must deploy or redeploy your application. Note that your Web Engagement Servers should be up and running during the deployment procedure.

6. **Create and Deploy Rules**
   Tool: Genesys Rules Authoring

   Description: You must create rules to optimize the event flow and create complex conditions to generate actionable events sent to the Genesys Solution. These rules link with the categories containing the business information. You can deploy rules only if the Web Engagement servers are started.

Application Development Tasks

You must complete the following steps to create a Genesys Web Engagement application:

1. Before developing an application, you must first install and configure Genesys Web Engagement and its components in a lab environment. See the Standalone Deployment Scenario for details and step-by-step instructions.
2. Creating an Application
3. Starting the Web Engagement Servers

4. Generating and Configuring the Instrumentation Script
5. Customizing an Application
    a. Creating Business Information
    b. Publishing the CEP Rule Templates
    c. Customizing the SCXML Strategies
    d. Customizing the Browser Tier Widgets
6. Deploying an Application
7. Creating a Rules Package
8. Testing with ZAP Proxy
9. Once you are satisfied with your application and are ready to deploy it to production, you should return to the Deployment Guide and deploy and configure the Web Engagement Cluster. See the Cluster Deployment Scenario for details and step-by-step instructions.

Creating an Application

You must create an application to run Genesys Web Engagement — see Application Development for details about the workflow for creating and deploying an application.

Complete the procedures on this page to create an application and then define its monitoring domains.

Creating a New Application

In this procedure you'll run the **create** script (**create.bat** on Windows and **create.sh** on Linux) to create your project structure. This script creates all the files required to run Genesys Web Engagement on your website.

Start

1. Navigate to the ***GWE_installation_directory*** and type the following command:

   ```
   create your_application_name.
   ```

End

Note: To request debug-level logs while this command is executed, use the **-v** parameter. For example:

```
create myApp -v
```

Result

A folder named ***your_application_name*** is created in ***GWE_installation_directory*****/apps**.

This folder contains all the materials used to deploy your application:

- **resources** contains the resources used by the app, including:
  - **_composer_project**, which contains all the SCXML default templates for the routing strategies and GRS rule template project. In addition, it contains the source code for the Browser Tier Widgets used for engagements.
  - **drl** contains your application's rules.
  - **dsl** contains your application's DSL.
  - The rest of the resources are widget-specific.

Next Steps

- Configuring the Instrumentation Script

Configuring the Instrumentation Script

The Tracker Application instrumentation script is a small piece of JavaScript code that you paste into your website to enable Web Engagement functionality. If you plan to use the Genesys Chat Widget or other Genesys Widgets, you must create your instrumentation using Genesys Widgets, in which case the Tracker Application provides built-in integration with Genesys Widgets.

> **Important**
>
> The rest of this article describes how to configure your instrumentation script. However, you must only use this technique if you are going to use the Tracker Application with "legacy" GWE Widgets. If you are using Genesys Widgets you must use the appropriate instrumentation script.

You typically add the instrumentation script to your site when you are ready to move your application to a production environment with a Web Engagement cluster. If you are working in a standalone deployment in a lab environment, you can use the default ZAP Proxy implementation to inject the instrumentation script into the pages of your web site on the fly.

You can complete the steps on this page to do the following:

1. Copy-paste basic instrumentation script.
2. Configure the script, if necessary for your solution.
3. Add the script to your website.

Copy-paste the Instrumentation Script

> **Important**
>
> This section is only for use in creating a standalone Tracker application that uses the native Web Engagement widgets. If you are using Genesys Widgets you must use the appropriate instrumentation script.

Start

1. Go to one of the chapters with the instrumentation script (such as Basic Instrumentation, Basic Configuration with the Chat JS Application, and Advanced Configuration chapters).
2. Copy instrumentation script and save it on your file system.

3. In the saved script:
   - Replace **server:port** with HTTP entry point to the Web Engagement Cluster (either Load Balancer or direct address of direct address of one of the servers). For example, `http://myserver.genesys.com:9081`
   - Replace **server:securePort** with HTTPS entry point to the Web Engagement Cluster (either Load Balancer or direct address of direct address of one of the servers). For example, `https://myserver.genesys.com:3214`
   - Replace GPE.min.js with other supported type of Tracker JS Application, if you need it

End

Next Steps

- You can configure your saved script.
- You can add the script to your website.

Configuring the Instrumentation Script

The Tracker Application activates the Monitoring and Notification functions in Genesys Web Engagement by inserting the **GTCJ.min.js** package into the page. This package includes jQuery, the Monitoring Agent, and the Notification Agent. The Tracker Application actually provides several packages that contain different functions and libraries. You can use these packages to enable different Web Engagement functionality on your website.

The table below shows the packages, in minified form, that are included with the Tracker Application.

Script	jQuery	Monitoring Agent	Notification Agent	Chat
GT.min.js	no	yes	no	no
GTJ.min.js	yes	yes	no	no
GTC.min.js	no	yes	yes	no
GTCJ.min.js	yes	yes	yes	no
GPE.min.js	yes	yes	yes	yes

Important

You must not make any changes to the scripts listed in the table above; any modifications will not be supported by Genesys. Please refer to the Genesys Web Engagement API Reference for information about the supported APIs.

The Tracker Application instrumentation script consists of two parts: configuration and script loader.

Script Loader

To load the Tracker Application, you just need to include the JavaScript in your web pages. This asynchronously loads the application, which means that it won't block other elements on your web pages from loading.

One solution for loading the script could be:

```
(function(gpe) {
    if (document.getElementById(gpe)) return;
    var s = document.createElement('script'); s.id = gpe;
    s.src = ('https:' == document.location.protocol ? 'https://<Web Engagement
Server>:<Secure Web Engagement Server Port>':
        'http://<Web Engagement Server>:<Web Engagement Server Port>') + '/server/resources/
js/build/GTCJ.min.js';
    (document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_gt');
```

> **Important**
>
> The script above uses the default "_gt" as the configuration global variable.

For more information about best practices for loading the script, see Adding the Instrumentation Script to Your Website.

Configuration

By default, the Tracker Application script uses the "_gt" global variable (you can change this in the script loader — see Changing the Global Configuration Variable for details) that must be initialized before the script loader is actually added to the page.

The following configuration options are available in the script:

Parameter	Required	Type	Default Value	Description	Example value
cookieExpires	no	number		Base expiration value is 3 years. With this option you are able to shift expiration, in seconds. Note, that this options does not overwrite expiration, it is shift it to base 3 years value.	Set expiration shifted to 17 seconds : "17", cookies will be set with 3 years + 17 seconds expiration time Set expiration to absolute value 60 seconds : "-94607940", which is -3*365*24*60*60+60 seconds
debug	no	boolean	false	Show Monitoring Agent debug information in the browser console.	true
debugCometD	no	boolean	false	Show CometD debug information in the browser console.	true
disableAutoSystemEvents	no	boolean	false	Disable automatic sending of the following system events: VisitStarted, PageEntered, PageExited.	true
disableIntegrationAgent	no	boolean	false	Supported starting with version 8.5.100.17. Disables the possibility of communication between TrackerJS script and other browser windows. **Note:** InTools functionality will not work in this case.	true
disableWebSockets	no	boolean	false	Disable websockets transport for the	true

Parameter	Required	Type	Default Value	Description	Example value
				notification agent. By default, the Notification Agent uses websocket transport when it is possible. Make sure that your load balancers support websocket connections; otherwise, disable it — Disabling Websocket CometD Transport.	
domainName	no	string	Second-level domain (SLD).	Name of the domain where the cookie is stored.	For the domain sub.genesys.com, the second-level domain is genesys.com
dslResource	no	string	-	The DSL resource location. If dslResource is not defined, then the DSL is not loaded.	http://genesyslab.com:9081/server/api/resources/v1/dsl/domain-model.xml
httpEndpoint	yes (if "httpsEndpoint" is undefined)	string	-	The URL of the Web Engagement Server.	http://genesyslab.com:9081
httpsEndpoint	yes (if "httpEndpoint" is undefined)	string	-	The secure URL of the Web Engagement Server.	https://genesyslab.com:8443
languageCode	no	string	en-US	Localization tag for language and region. Used for categorization.	en-US
name	no	string	-	Name of the application. This option is a part of the cloud multi-tenant, multi-domain system. Currently not used.	genesyslab

Parameter	Required	Type	Default Value	Description	Example value
page	no	object	-	Sets the page configuration for events. In some cases, you might want to set a parameter and have the value persist across multiple push events. To override the page url of each event with your own custom url, you can either set the new url on each push command, or you can use current option. **Note:** This option should only be used with Single Page Applications.	`_gt.push(['config', {<br class="mw_emptyline">page: {<br class="mw_emptyline">url: 'http://example.com/my-page-url?id=1', title: 'My Page Title'<br class="mw_emptyline">}<br class="mw_emptyline">}]);`
page:title	no	string	document.title	The title of the current page (this title is used in the PageEntered event by default).	
page:url	no	string	window.location.href	The URL of the current page. This option is used for all subsequent events sent from the page.	
preventIframeMonitoring	no	boolean	false	If preventIframeMonitoring is true, the Monitoring Agent does not generate system and business events if the agent is loaded in an iframe. See preventIframeMonitoring	true

Parameter	Required	Type	Default Value	Description	Example value
				for details.	
skipCategories	no	boolean	false	Do not include category information with server response to initial page request. This option can be used when a website does not need to use categories.	true

Basic Configuration

Basic configuration is the default Tracking functionality:

```
var _gt = window._gt || [];
_gt.push(['config', {
        dslResource:    ('https:' == document.location.protocol ? 'https://server:securePort'
:
                        'http://server:port') + '/server/api/resources/v1/dsl/domain-
model.xml',
        httpEndpoint: 'http://server:port',
        httpsEndpoint: 'https://server:securePort'
}]);

(function(gpe) {
        if (document.getElementById(gpe)) return;
        var s = document.createElement('script'); s.id = gpe;
        s.src = (   'https:' == document.location.protocol ? 'https://server:securePort' :
                    'http://server:port') + '/server/resources/js/build/GTCJ.min.js';
        (document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_gt');
```

This snippet represents the minimum configuration needed to track a page asynchronously. The `_gt` (Genesys Tracker) object is what makes the asynchronous syntax possible. It acts as a queue, which is a first-in, first-out data structure that collects API calls until Genesys Web Engagement is ready to execute them. To add something to the queue, you can use the `_gt.push` method. See the Monitoring JS API for more information.

Basic Configuration with the Chat JS Application

To add chat functionality to the basic configuration by loading the Chat JS Application, your script should now look something like this:

```
var _gt = window._gt || [];
_gt.push(['config', {
    dslResource:    ('https:' == document.location.protocol ? 'https://server:securePort' :
                    'http://server:port') + '/server/api/resources/v1/dsl/domain-model.xml',
    httpEndpoint: 'http://server:port',
    httpsEndpoint: 'https://server:securePort'
    }]);

var _gwc = {
    widgetUrl: ('https:' == document.location.protocol ? 'https://server:securePort' :
                    'http://server:port') + '/server/api/resources/v1/chatWidget.html'
};

(function(gpe) {
    if (document.getElementById(gpe)) return;
    var s = document.createElement('script'); s.id = gpe;
    s.src = (   'https:' == document.location.protocol ? 'https://server:securePort' :
                'http://server:port') + '/server/resources/js/build/GPE.min.js';
    (document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_gt');
```

Advanced Configuration

The snippet below shows the instrumentation script with extended configuration (refer to the configuration options table for details):

```
var _gt = _gt || [];
_gt.push(['config', {
    name:              'demo',
    domainName:        'localhost',
    languageCode:      'en-US',
    dslResource:     ('https:' == document.location.protocol ? 'https://server:securePort':
        'http://server:port') + '/server/api/resources/v1/dsl/domain-model.xml',
    httpEndpoint: 'http://server:port',
    httpsEndpoint: 'https://server:securePort'
    languageCode:      'en-US',
    debug:             true,
    debugCometD:        true,
    preventIframeMonitoring: true,
}]);

(function(gpe) {
    if (document.getElementById(gpe)) return;
    var s = document.createElement('script'); s.id = gpe;
    s.src = (   'https:' == document.location.protocol ? 'https://server:securePort' :
        'http://server:port') + '/server/resources/js/build/GTCJ.min.js';
    (document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_gt');
```

preventIframeMonitoring

Some websites have iframe (or frame) elements on the page. If a website is instrumented so that the Monitoring Agent is loaded on all web pages (even in an iframe), the agent generates events for all pages, including iframes. For example, this means that a page with an iframe generates two PageEntered events, one for the main page and one for the iframe.

To prevent this, you can use a special initialization parameter, `preventIframeMonitoring`. This parameter is optional and has a default value of `false`. If `true`, the Monitoring Agent does not generate system and business events if it is loaded in an iframe.

Changing the Global Configuration Variable

You can change the global configuration variable for the Tracker Application by using the `data-gpe-var` attribute. For example:

```
(function(gpe) {
    if (document.getElementById(gpe)) return;
    var s = document.createElement('script'); s.id = gpe;
    s.src = ('https:' == document.location.protocol ? 'https://server:securePort':
        'http://server:port') + '/server/resources/js/build/GTCJ.min.js';
    s.setAttribute('data-gpe-var', gpe); // set global variable name for Tracker Application
    (document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_myVariable');
```

In the example above global variable "_myVariable" is now used instead of "_gt".

Providing an External jQuery Library

If you already have a jQuery library on your website, you can reduce the size of the Genesys Web Engagement JavaScript files by using the packages without jQuery (**GT.min.js** or **GTC.min.js**). In this case, make sure that jQuery is available on your site through the global variable `window.jQuery` and that jQuery is loaded before the Genesys Tracker Application.

If the jQuery library is present on some pages and not others, you must insert the following snippet of code before the instrumentation script:

```
<script>
window.jQuery || document.write("<script src='http://code.jquery.com/
jquery-1.11.0.min.js'>\x3C/script>")
</script>
```

Disabling Websocket CometD Transport

To disable websockets CometD transport, use the **transports** option in your instrumentation script:

```
_gt.push(['config', {
    disableWebSockets: true,
    ....
}]);
```

Next Steps

- When you are satisfied with your script configuration, you can move on to either Adding the Instrumentation Script to Your Website or Customizing an Application (if you configured the script so it can be used with the ZAP Proxy).

Adding the Instrumentation Script to Your Website

To add the instrumentation script, you need to have access to the source code for your website. If you already have an older version of the instrumentation script on your site, make sure you remove it from each page before you add the new one. If you have customizations you want to add back to your pages after you add the new snippet, you can use a text or HTML editor to open and save a copy of each file.

The instrumentation script is loaded asynchronously. One of the main advantages of the asynchronous script is that you can position it at the top of the HTML document. This increases the likelihood that the tracking beacon will be sent before the user leaves the page. Genesys recommends placing the script at the bottom of the **<head>** section for best performance.

For the best performance across all browsers, Genesys recommends that you position other scripts in your site either before the instrumentation script in the **<head>** section or after both the instrumentation script and all page content (at the bottom of the HTML body).

Make sure that the document type is defined in the head of each of your web pages. If it is not defined, Genesys Web Engagement will not work on your website.

```
<!DOCTYPE html>
```

Prerequisites

- You removed any older versions of the instrumentation script from your site.
- You saved the instrumentation script.

Start

1. Select and copy the script from your file.
2. Paste the script at the bottom of the **<head>** section of your web pages:
   - You can do this manually on each web page that you want to monitor.
   - You can do this in the header template of your website, if you have one.
3. If your website includes additional scripts, do one of the following to optimize performance:
   - Place your scripts above the instrumentation script in the **<head>** section.
   - Make sure your scripts are located after the webpage contents (at the bottom of the **body** section).

End

Next Steps

- Starting the Web Engagement Server

Starting the Web Engagement Server

You can start the Web Engagement Server from Genesys Administrator, from the start script, or as a Windows service.

Start

To start your server from Genesys Administrator:

1. Navigate to **Provisioning > Environment > Applications**.
2. Select the Web Engagement Server.
3. Click **Start applications** in the **Runtime** panel.

To start your server using the provided **start** script (**server.bat** on Windows and **server.sh** on Linux):

1. Navigate to the Web Engagement installation directory and launch a console window.
   - For Windows, type: `server.bat`
   - For Linux, type: `server.sh`

End

The Web Engagement Server is started.

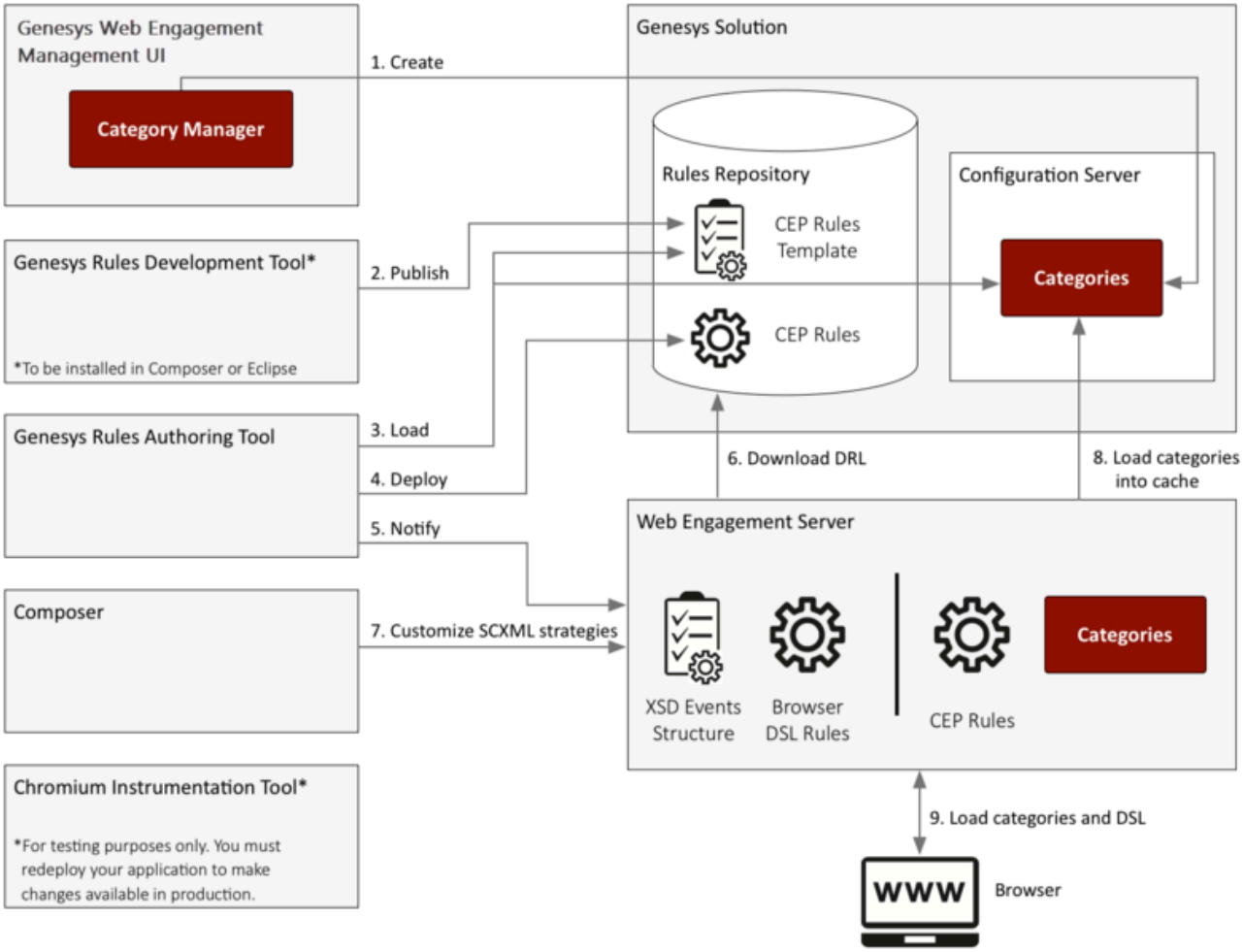
Next Steps

- Customizing an Application

Customizing an Application

When you develop a Web Engagement application, you start by creating your application with the script tools which generates default SCXML strategies, rule templates, and DSL code. You can customize all of this material through specific tools.

The diagram below shows where you can customize the Web Engagement data used by your application.



1. If you are following the Simple Engagement Model, you **create** categorization information with the Categories interface in the Genesys Web Engagement Management interface. This information is added to Configuration Server and retrieved by the Web Engagement Server. When the Web Engagement Server receives a browser request, it checks the category information. If you are following the Advanced Engagement Model, you **create** business events by modifying the DSL using the InTools application.

2. You must **publish** the CEP rule template associated with your engagement model. You can modify this template before you publish it.
3. The Genesys Rules Authoring Tool (GRAT) **loads** the CEP Rule template and allows you to create a package of CEP rules based on your categories (Simple Engagement Model) or on your business events (Advanced Engagement Model).
4. If your Web Engagement servers are started, you can **deploy** your application and deploy rules with GRAT.
5. GRAT **notifies** the Web Engagement Server that rules are available in the Rules repository.
6. The Web Engagement Server **downloads** the rules. You can use the InTools application to customize your DSL.
7. You can **customize** the SCXML strategies available in the **_composer-projects** directory located in ***Web Engagement installation directory*/apps/*application_name*/resources**. See Customizing the SCXML Strategies for details. At this point you can also customize the various Browser Tier Widgets.
8. When a browser submits a request to the Web Engagement Server, the Web Engagement Server **loads** the categories into the cache.
9. The user's web browser **loads** the updates.

Creating Business Information

You must create business information for your application following either the Simple Engagement Model, the Advanced Engagement Model, or a combination of both.

- The Simple Engagement Model derives categories from the content of the System events. With this model, you do not need to create Business events; instead, you create rules and category information based on the available out-of-the-box system events.
- The Advanced Engagement Model uses Business events defined in the Browser Tier Domain Specific Language (DSL) to create event-related rules. Once the business event is generated by the DSL, all the event attributes are available for complex event processing and for use by the SCXML strategies.

Simple Engagement Model

Overview

The Simple Engagement Model is a simple solution to add Web Engagement to your website with limited effort.

You can use the Web Engagement Management interface to define, in a few clicks, Web Engagement categories that contain business information related to URL or web page titles. These categories are used in the CEP rule templates, which provide rules that define when to submit actionable events to Web Engagement — this is what starts the engagement process.

For example, lets look at Solutions on the Genesys website. In this scenario, you can define a Solution category associated with the `http://www.genesys.com/solutions` page and several or all solution sub-pages, such as `http://www.genesys.com/solutions/cloud` or `http://www.genesys.com/solutions/enterprise-workload-management`.

- To associate the category with all the pages containing the "solutions" string in the URL, you can create the "solutions" tag. This tag defines the "solutions" string as a plain text expression to search in the events triggered by the visitor browsers.
- To set up a specific list of sub-pages for the Solutions category, you can create a tag for each sub-page:
  - The "cloud" tag, which defines the "cloud" string as the plain text expression to search in the events triggered by the visitor browsers.
  - The "enterprise-workload" tag, which defines the "enterprise-workload-management" string as the plain text expression to search in the events triggered by the visitor browsers.

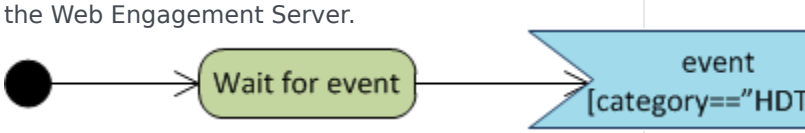
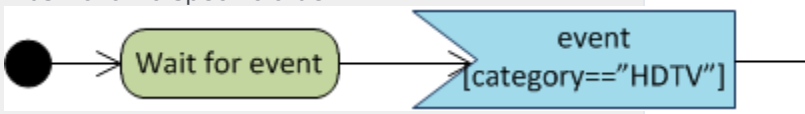
Now your rules can use this category to match solution-related pages.

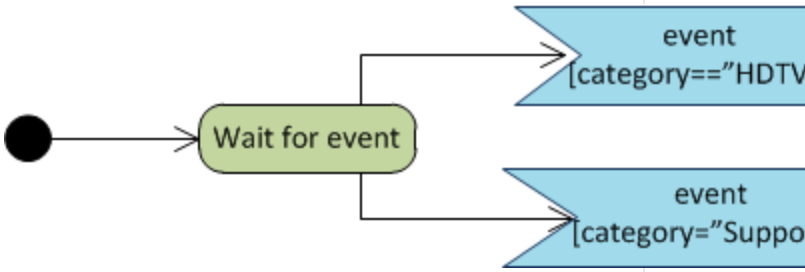
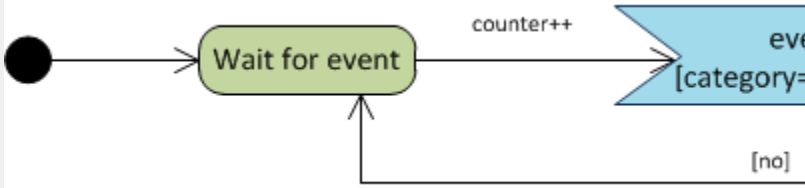
The templates for category-based rules define how to process events sent from the Web Engagement Server. They define both the type of events to take into account and the action to perform. The Genesys Rules Authoring Tool loads the template and uses its content to help you define rules. These templates are created with your application and can be modified with the Genesys Rules Development Plug-in (in Composer or in Eclipse).

Default Rule Templates

The default templates for the Simple Engagement Model define how to process events sent from the Web Engagement Server. They define both the type of events and the action to perform. Later, you'll use the Genesys Rules Authoring Tool to create rules based on these templates.

Singleton	
Description	The template receives each single event as a formal parameter. If the event's value matches the

	right category, then the actionable event is sent to the Web Engagement Server.  GWM single.png
Expression Example	**When** page transition event occurs that belongs to category $category **Then** generate actionable event
Sequence	
Description	This template analyses the event stream received from the categorization engine and builds the sequence of events by category values. As soon as the event sequence is completed, the actionable event is submitted. Note that the event sequence must follow a specific order.  Click to enlarge.
Expression Example	**When** page transition event occurs that belongs to category `$category1` save as `$event1` **and** event following `$event1` with category `$category2` save as `$event2` (...) **and** event following $event^{n-1} with category $category^{n} save as $event^{n} **Then** generate actionable event based on $event^{n}
Set	
Description	This template analyses the event stream received from the categorization engine and collects the events by category values. As soon as the event set is completed, the actionable event is submitted. If you use this template, the event order is not taken into account.

	 GWM Set.png
Expressions	**When** (page transition event occurs that belongs to category `$category1` **and** page transition event occurs that belongs to category `$category2`) **or** (page transition event occurs that belongs to category `$category2` **and** page transition event occurs that belongs to category `$category1`) **Then** generate actionable event
Counter	
Description	This template analyses the event stream received from the categorization engine and counts events which occur for a given category. As soon as the counter is reached, the actionable actionable event is submitted.  GWM Counter.png
Expressions	**When** Category `$category` counts `$count` times **Then** generate actionable event

Implementing the Simple Engagement Model

You can add and remove categories for Web Engagement through the Category interface in the Web Engagement Management interface. You create these categories during the Application Development process if you use the Simple Engagement Model when you Create Business Information.

Next Steps

1. Make sure the CEP Rule Templates are ready. See Publishing the CEP Rule Templates for details.
2. Finish any customizations to the SCXML strategies or Browser Tier Widgets.
3. Continue on with the Application Development Tasks.

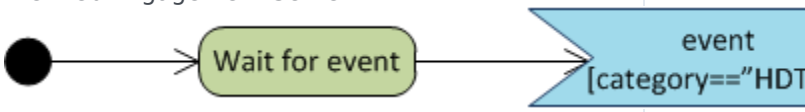
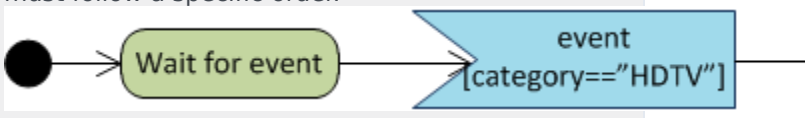
Advanced Engagement Model

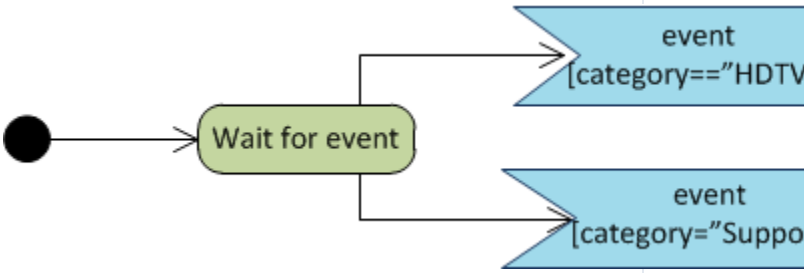
Overview

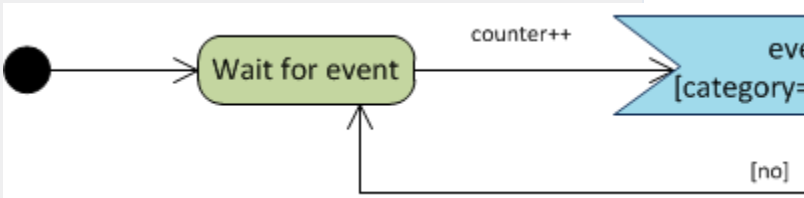
The Advanced Engagement Model enables customization based on Business events (read more about how the events are structured here). In Web Engagement 8.5.1, the default DSL contains the `Timeout-30` event and a sample of a `Search` event. To customize the Advanced Engagement Model, you must first define your own events using the DSL, which is loaded in the Browser Tier Agents. Then, you can use the rule templates to create rules based on these events.

Default Rule Templates

The default templates for the Advanced Engagement Model define how to process events sent from the Web Engagement Server. They define both the type of events and the action to perform. Later, you'll use the Genesys Rules Authoring Tool to create rules based on these templates.

Singleton	
Description	The template receives each single event as a formal parameter. If the event's value matches the right category, then the actionable event is sent to the Web Engagement Server.  GWM single.png
Expression Example	**When** page transition event occurs that belongs to category $category **Then** generate actionable event
Sequence	
Description	This template analyses the event stream received from the categorization engine and builds the sequence of events by category values. As soon as the event sequence is completed, the actionable event is submitted. Note that the event sequence must follow a specific order. 

	Click to enlarge.
Expression Example	**When** page transition event occurs that belongs to category `$category1` save as `$event1` **and** event following `$event1` with category `$category2` save as `$event2` (...) **and** event following $event^{n-1} with category $category^{n} save as $event^{n} **Then** generate actionable event based on $event^{n}
Set	
Description	This template analyses the event stream received from the categorization engine and collects the events by category values. As soon as the event set is completed, the actionable event is submitted. If you use this template, the event order is not taken into account.  GWM Set.png
Expressions	**When** (page transition event occurs that belongs to category `$category1` **and** page transition event occurs that belongs to category `$category2`) **or** (page transition event occurs that belongs to category `$category2` **and** page transition event occurs that belongs to category `$category1`) **Then**

	generate actionable event
Counter	
Description	This template analyses the event stream received from the categorization engine and counts events which occur for a given category. As soon as the counter is reached, the actionable actionable event is submitted.  GWM Counter.png
Expressions	**When** Category `$category` counts `$count` times **Then** generate actionable event
Search	
Description	The actionable event is submitted if a `Search` event occurs.
Expressions	**When** event with name `Search` save as `$event1` **Then** generate actionable event based on `$event1`
Timeout	
Description	The actionable event is submitted if a `Timeout` event occurs.
Expressions	**When** event with name `Timeout` save as `$event1` **Then** generate actionable event based on `$event1`

Implementing the Advanced Engagement Model

Complete the steps below to implement the Simple Engagement Model:

1. Business Events Overview
2. Create Business Events by Customizing the DSL File
3. Optionally, you can Create Business Events by Using the Monitoring Agent API.

Business Events Overview

When you create an application, a set of Domain Specific Language (DSL) files that are used by your application is also created. These files are defined in the **apps*Your application name*\resources\dsl** directory. You can use the DSL to define Business events (read about the structure of these events here) that are specific to your solution needs.

Default domain-model.xml

The **domain-model.xml** is the main default DSL file for your application:

```
<?xml version="1.0" encoding="utf-8" ?>
<properties>
    <events>
        <!-- Add your code here
        <event id="" name="">
        </event>
        -->

        <!-- This is template for your search event -->
        <!--
        <event id="Search" name="Search">
            <trigger name="SearchTrigger" element="" action="click" url="" count="1" />
            <val name="searchString"       value="" />
        </event>
        -->
        <event id="Timeout-30" name="Timeout-30" condition=""
postcondition="document.hasFocus() === true">
            <trigger name="TimeoutTrigger" element="" action="timer:30000" type="timeout"
url="" count="1" />
        </event>

    </events>

</properties>
```

By using the **<event>** element, you can create as many business events as you need. These events can be tied to the HTML components of your page and can have the same name, as long as they have different identifiers (these identifiers must be unique across the DSL file, to make a distinction between the events sent by the browser). It can be useful to associate several HTML components with the same event if these HTML components have the same function. For instance, you can define several events associated with a search feature and give all these events the same name: "Search".

For each event, you can define triggers which describe the condition to match in order to submit the event:

- Triggers can implement timeouts.
- Triggers can be associated with DOM events.
- You can define several triggers for the same event.

Each trigger should have an `element` attribute that specifies the document's DOM element to attach the trigger to, and the `action` attribute, which species the DOM event to track.

You can specify standard DOM events for the `action`:

- Browser Events
- Document Loading
- Keyboard Events
- Mouse Events
- Form Events

In addition to the standard DOM events, the DSL supports the following two values: `timer` and `enterpress`.

The following example generates a "Search" event if the visitor does a site search. The "searchString" value is the string entered in the "INPUT.search-submit" form.

```
<event id="SearchEventClick" name="Search">
          <trigger name="SearchTrigger" element="INPUT.search-submit" action="click" url=""
count="1" />
          <val name="searchString" value="INPUT.search-submit" />
</event>
```

If the DSL uses the optional `condition` attribute, the event's triggers are installed on the page if the condition evaluates to true. The following example creates a Business event with a time that can be triggered only if the text inside the <h1> tag is "Compare":

```
<event id="InactivityTimeout4CompareProductsEvent" name="InactivityTimeout4CompareProducts"
condition="$('h1').text() == 'Compare'">
           <trigger name="InactivityTimeout4CompareProductsTrigger" element=""
action="timer:10000"
           type="timeout"  url="http://www.MySite.com/site/olspage.jsp" count="1"/>
</event>
```

If the DSL uses an optional `postcondition` attribute, this can manage how an event is generated by checking a condition after the actions are completed. The following example creates a Business event timeout by timer if a page is in focus. In this case, the event does not generate if the page is opened in the background:

```
<event id="TimeoutEvent10" name="Timeout-10" condition="" postcondition="document.hasFocus()
=== true">
         <trigger name="TimeoutTrigger" element="" action="timer:10000" type="timeout" url=""
count="1" />
</event>
```

A DSL `trigger` can use the `type` attribute. This can have a value of either `timeout` or `nomove`, which specifies how the timer action works. If the `type` is `timeout`, then the timer interval begins after the page is loaded. If the `type` is `nomove`, then the timer resets each time the user moves the mouse.

You can also apply the optional `url` attribute. This attribute defines the URL of the specific page that raises the Business event. The Business event is not submitted if the current document's URL does not match the URL parameter.

Finally, you can apply the optional `count` attribute. This attribute specifies how many times the

trigger needs to be matched before the event is generated and sent to the Web Engagement Server.

For more information about the DSL elements, see the Business Events DSL.

Creating Business Events by Customizing the DSL File

You can edit the **apps*Your application name*\resources\dsl\domain-model.xml** and add a list of events, with specific conditions, related to your web pages' content.

> **Important**
>
> Genesys recommends that you use the InTools application to help you modify your DSL.

The default **domain-model.xml** file includes two sample events to help you get started with your DSL customizations: **Timeout-30** and a prototype of the **Search** event (commented out by default). The following sections show you how you can customize these events to work on your website.

Using the Search Event Template

By default, the **domain-model.xml** file contains commented code that you can implement to trigger a business event when a visitor tries to search for something on your website. Complete the following steps to customize the Search event for your website.

Start

1. Remove the comment characters that wrap around the event: <!– and -->. The event should look like the following:

```
<event id="Search" name="Search">
    <trigger name="SearchTrigger" element="" action="click" url="" count="1" />
    <val name="searchString"       value="" />
</event>
```

2. Set the **element** attribute to the jQuery selector that triggers a search. For example, we have an input (`id="search"`) with a submit button (`id="search-submit"`).

```
<event id="Search" name="Search">
    <trigger name="SearchTrigger" element="#search-submit" action="click" url=""
count="1" />
    <val name="searchString" value="" />
</event>
```

3. Set the **value** attribute to the script to retrieve the search string. For example, our input id of "search".

```
<event id="Search" name="Search">
    <trigger name="SearchTrigger" element="#search-submit" action="click" url=""
count="1" />
    <val name="searchString" value="$(#search).val()" />
</event>
```

Now the search event is triggered when a visitor clicks the **search-submit** button.

End

Using the Timeout Events

By default, the **domain-model.xml** file contains the **timeout-30** timeout event.

```
<event id="Timeout-30" name="Timeout-30" condition="" postcondition="document.hasFocus() ===
true">
    <trigger name="TimeoutTrigger" element="" action="timer:30000" type="timeout" url=""
count="1" />
</event>
```

You can customize this event or disable it to suit your business needs. By default, this event is triggered with a 30-second delay after the tracking script is initialized on the page. The only difference between the events is the **action** attribute, which defines the timeout in milliseconds.

The default timeout event has the **postcondition** attribute set to `"document.hasFocus() === true"`, which checks whether the focus is on the current page. The timeout event is only triggered if the **postcondition** returns true.

Creating Business Events by Using the Monitoring Agent API

You can also use the Monitoring JS API, which allows you to submit events and data from the HTML source code.

In this case, you can use the `_gt.push()` method which allows you to decide when events should be submitted and which data they generate, directly from your web pages. See Monitoring JS API Reference for further details.

You should also consider using the API when you have more complex logic that can't be handled by DSL alone. For an example, see How To — Enable a trigger after another trigger.

Next Steps

1. Make sure the CEP Rule Templates are ready. See Publishing the CEP Rule Templates for details.
2. Finish any customizations to the SCXML strategies or Browser Tier Widgets.
3. Continue on with the Application Development Tasks.

Publishing the CEP Rule Templates

After you create business information by following either the Simple Engagement Model or the Advanced Engagement Model, you can begin working with the CEP Rule Templates.

Even if you do not plan to customize the CEP rule templates, you still need to import, configure, and publish them in the rules repository so that they are available when you begin creating your rules. You can do this in two different ways:

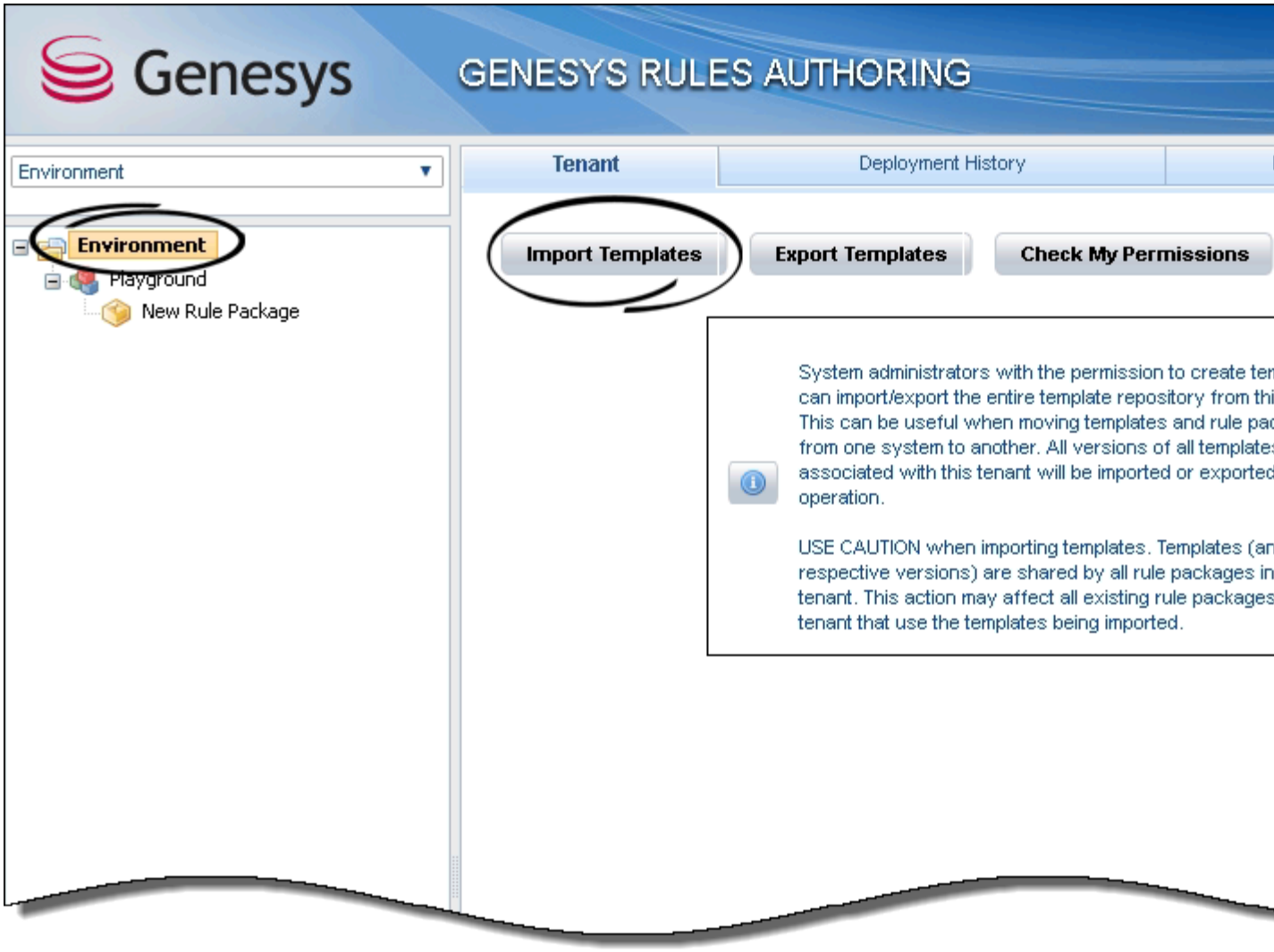
- Import the default CEP rule template into the Genesys Rules Authoring Tool (GRAT) and then use it as is. This is known as the simple mode of default CEP rule template publishing.
- Use the Genesys Rules Development Tool (GRDT) to import the default CEP rule template, then modify it and publish to the GRAT repository. This is known as the advanced mode of CEP rule template publishing.

Simple mode of default CEP rule template publishing

If you do not plan to introduce new business events and will use only those available in the out-of-the-box DSL file—or if you just want to get off to a quick start with the default CEP rule template, without using GRDT—you can use the simple mode of default CEP rule template publishing, which only requires GRAT.

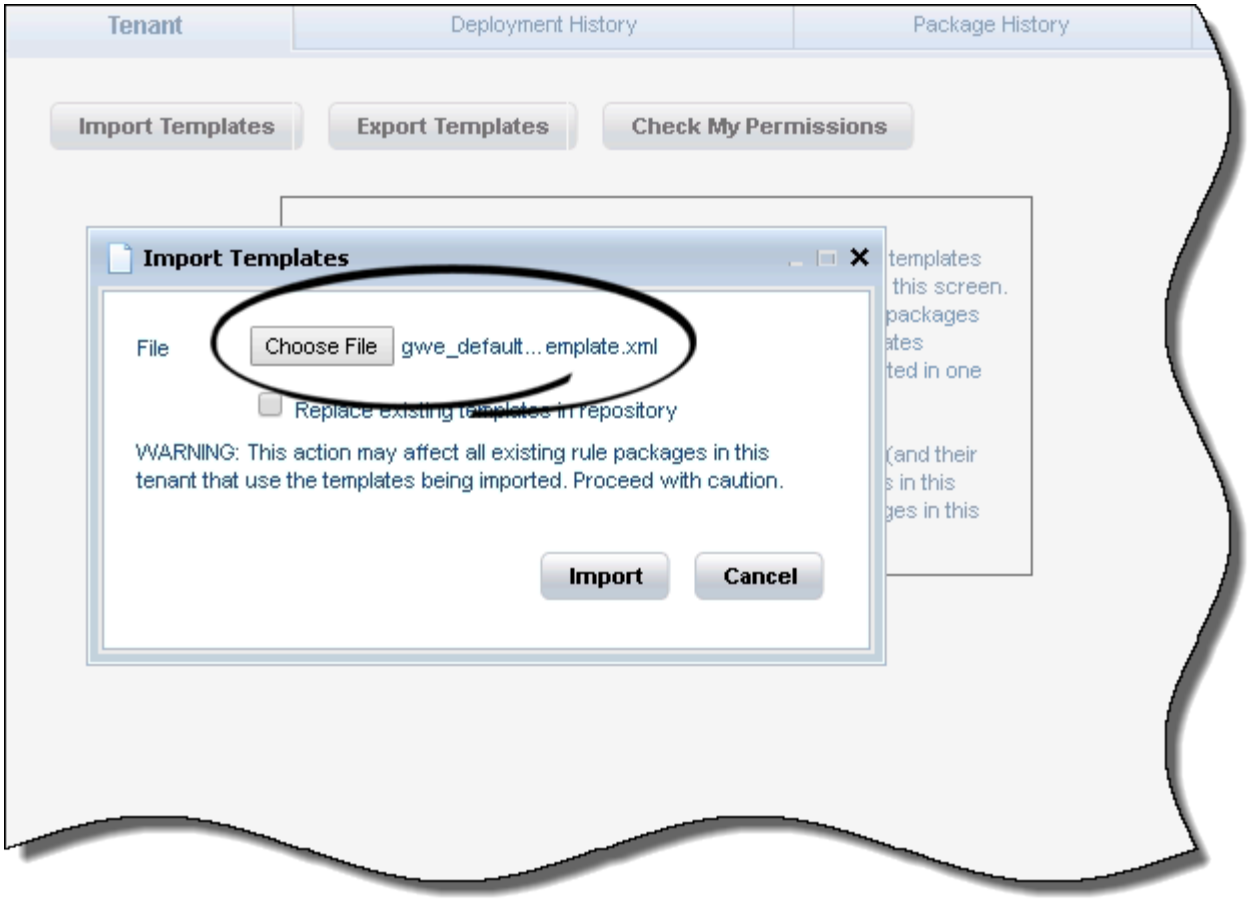
Here's how:

1. Verify that **gwe_default_grat_template.xml** is present in ***Web Engagement installation dir*\apps*your application*\resources_composer-projects\WebEngagement_CEPRule_Templates\import**. This file contains the default Web Engagement CEP rule templates.
2. Open GRAT and select the **Environment** element from the left pane. The **Import Templates** button appears in the right pane.



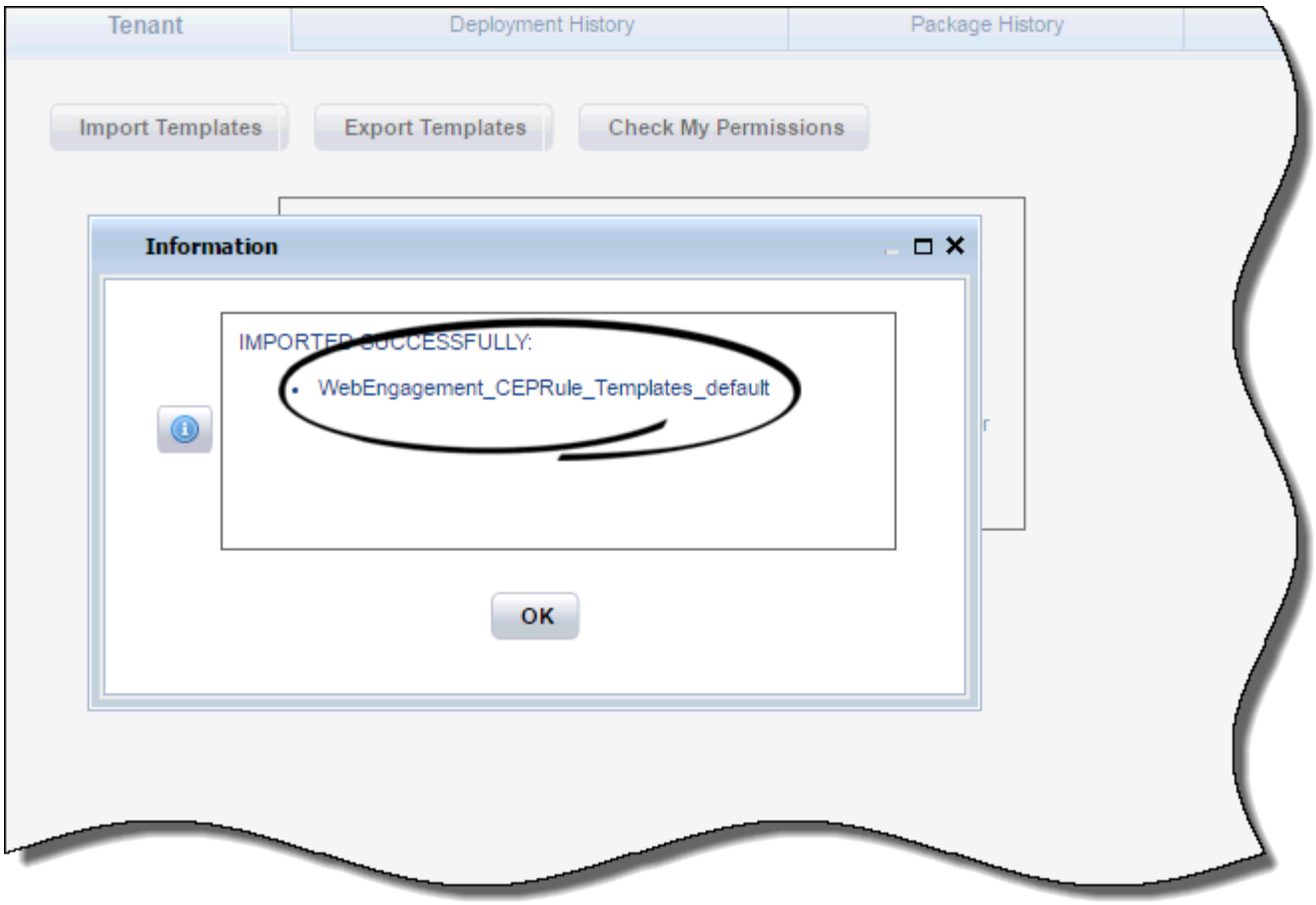
GRAT Import Templates Button

3. Click **Import Templates**.
4. Browse to **gwe_default_grat_template.xml**, then then click the **Import** button:



GRAT Import Templates Dialog

5. If the template file was successfully imported, you will see a confirmation dialog:



GRAT Confirmation Dialog

6. You can now create your own rule package, based on the imported CEP rule template:

New Rule Package

Advanced mode of CEP rule template publishing

To use the advanced mode, do this:

1. Read the overview information about the rule templates.
2. Importing the CEP Rule Templates in GRDT.
3. Configuring the CEP Rule Templates.
4. If necessary, you can Customize the CEP Rule Templates.
5. Publishing the CEP Rule Templates in the Rules Repository.

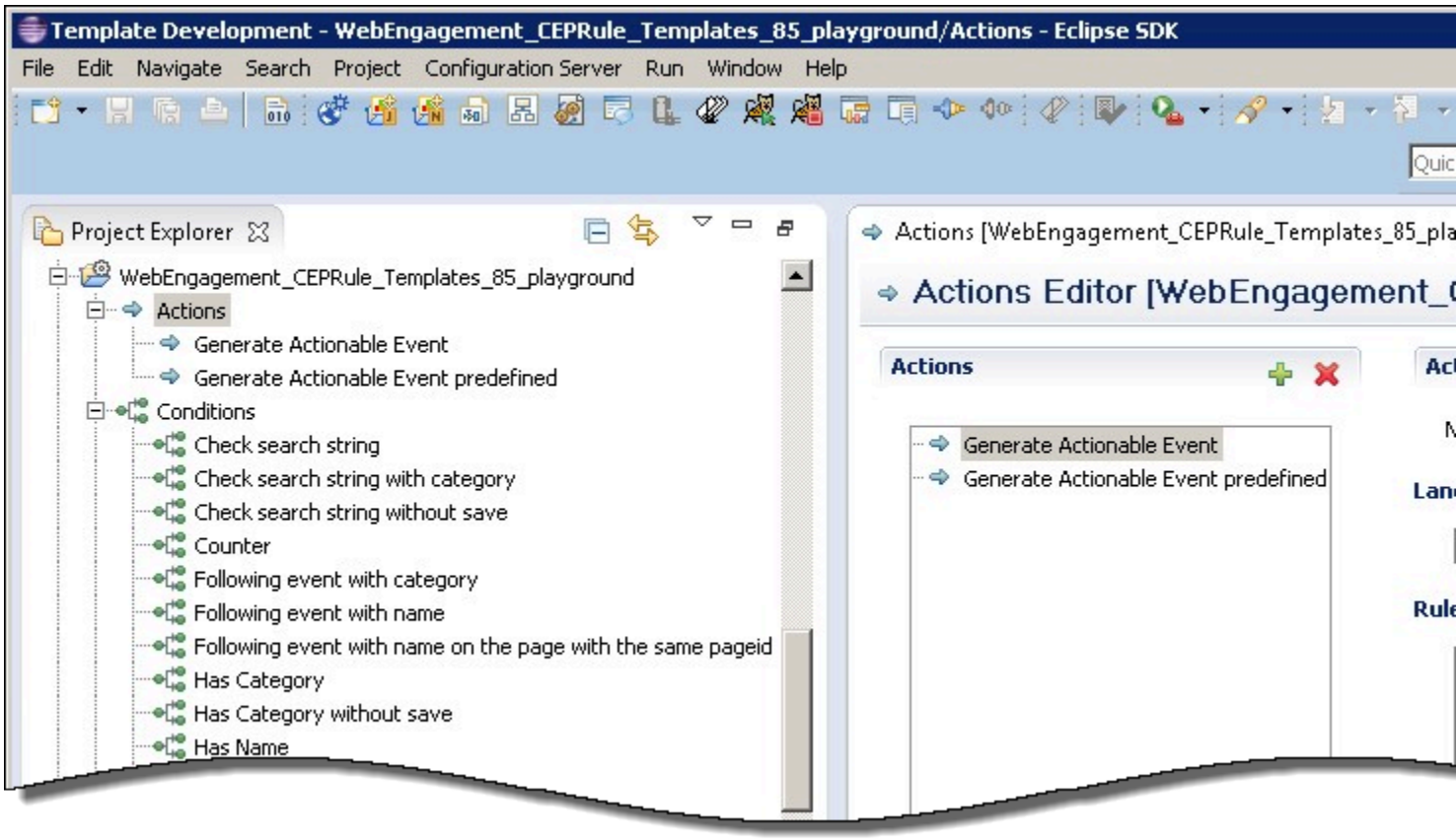
Overview

The Complex Event Processing (CEP) Rule Templates define the actions and conditions you can use when you create your business rules in Genesys Rules Authoring Tool.

You use the Genesys Rules Authoring Tool (GRAT) to develop, author, and evaluate these business rules. A business rule is a piece of logic defined by a business analyst. These rules are evaluated in a Rules Engine based upon requests received from client applications such as Genesys Web Engagement. A newly created Web Engagement application contains a pre-defined CEP (Complex Event Processing) template. This template type enables rule developers to build templates that rule authors then use to create rules and packages. These rules use customized event types and rule conditions and actions. Each rule condition and action includes the plain-language label that the business rules author will see, as well as the rule language mapping that defines how the underlying data will be retrieved or updated.

By default, your newly created Web Engagement application contains the following CEP Rule Template:

- **\apps*application name*\resources_composer-projects\WebEngagement_CEPRule_Templates** includes a GRDT-based project with CEP templates.



CEP rule template in Composer

In order to use these templates to define rules, you must first publish them.

Before you publish the templates, you can edit them to suit your business needs using the the Genesys Rules Development Tool. For more information about rule templates, refer to the Genesys Rules System documentation.

> **Important**
> Note that if you customize your rule templates, you must republish them.

Actions

The list of actions available in the template is listed in **WebEngagement_CEPRule_Templates > Actions**. You can edit, add, or remove these actions. In the Genesys Rules Authoring Tool (GRAT), when you create a rule based on the template, you can add an action by clicking **Add action**; GRAT displays all the actions defined in the template. You'll see how actions are implemented once you start creating rules. The default actions are:

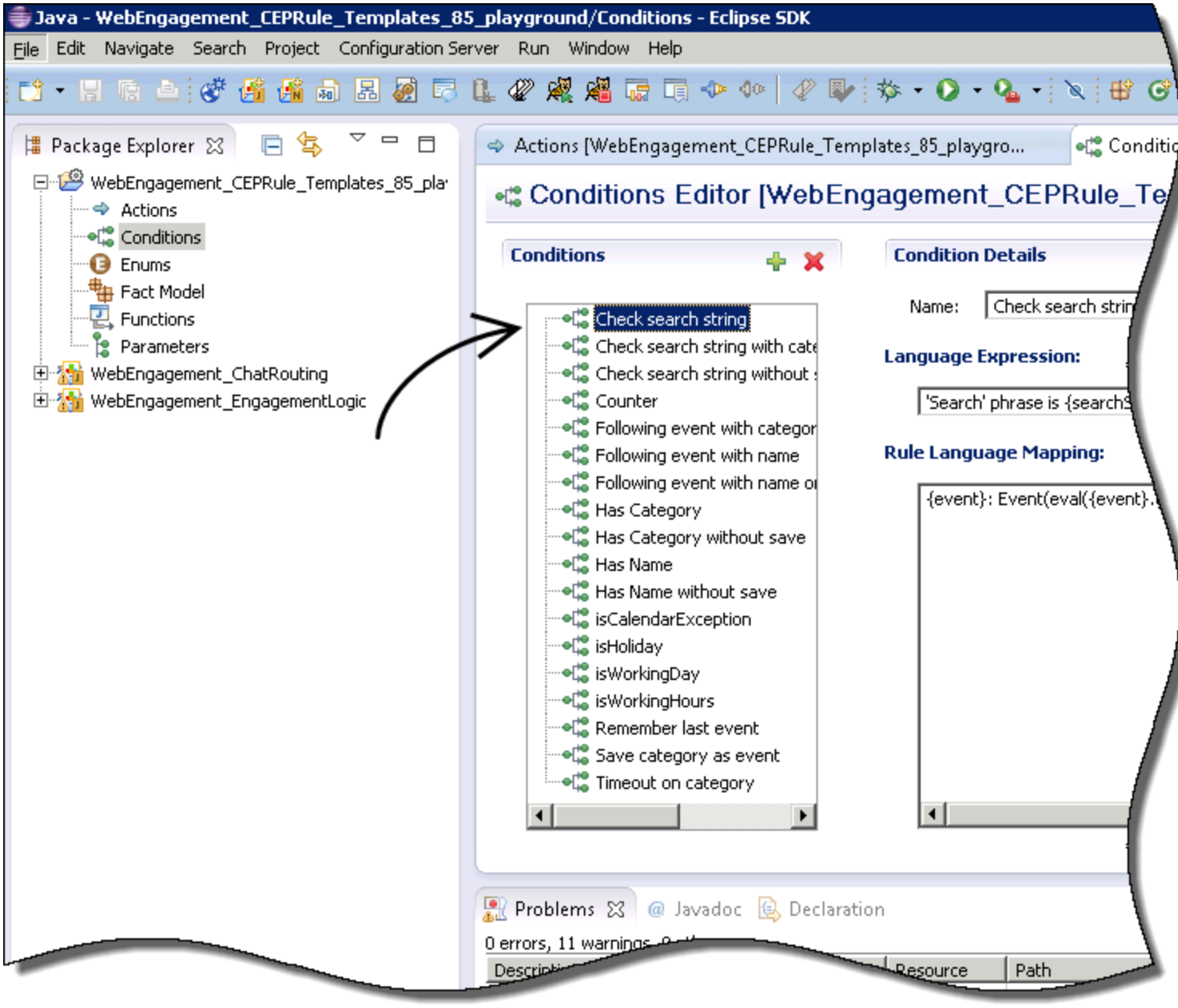
- Generate Actionable Event
- Generate Actionable Event Predefined

Enums

The enumerations available in the template are listed in **WebEngagement_CEPRule_Templates > Enums**. You can edit, add, or remove these enumerations. When you create a rule based on the template, you can specify a **Phase** by clicking **Add Linear Rule**; GRAT displays all the enumerates available in the template. In the default template, no specific enumeration is available.

Conditions

The conditions are listed in **WebEngagement_CEPRule_Templates > Conditions**.



List of conditions in the CEP rule template.

You can edit, add, or remove these conditions. Each condition associates a name with an expression. When you create a rule based on the template, you can add one or more condition to this rule by clicking **Add condition**; GRAT displays all the condition expressions available in the template. For complex templates, you need several conditions to implement a rule.

Condition Details

Condition Name	Expression	Condition details
Check search string	event searches {searchString}	Returns true if the event Search occurs and if the {searchString}

Condition Name	Expression	Condition details
		label is found, this event's result is saved in the {event} label.
Following event with category	AND event following {prevEvent} with category {category} save as {event}	If the event follows {prevEvent} and contains the {category} label, this event's result is saved in the {event} label.
Following event with name	AND event following {prevEvent} with name {eventName} save as {event}	If the {eventName} follows {prevEvent} in parameter, this event's result is saved in the {event} label.
Has Category	page transition event occurs that belongs to category {category} save as {event}	If the event is a page transition for the given category, this event's result is saved in the {event} label.
Has Category without save	page transition event occurs that belongs to category {category}	Returns true if the event is a transition to the given category's page.
Has Name	event with name {eventName} save as {event}	If the {eventName} occurs, this event's result is saved in the {event} label.
Has Name without save	AND event with name {eventName}	Returns true if {eventName} occurs.
Remember last event	Precondition: save last event	Saves the last event.
Save category as event	category is {category} save as {event}	If the event contains the given category, this event's result is saved in the {event} label.
Timeout on category	Timeout event occurs with category {category}	Returns true if the Timeout event occurs for the given category.

Importing the CEP Rule Templates in GRDT

Complete this procedure to import the CEP rule templates in the Genesys Rules Development Tool. Even if you do not plan to customize the templates, your rule template must be published in the Rules System Repository before you try to create rules.

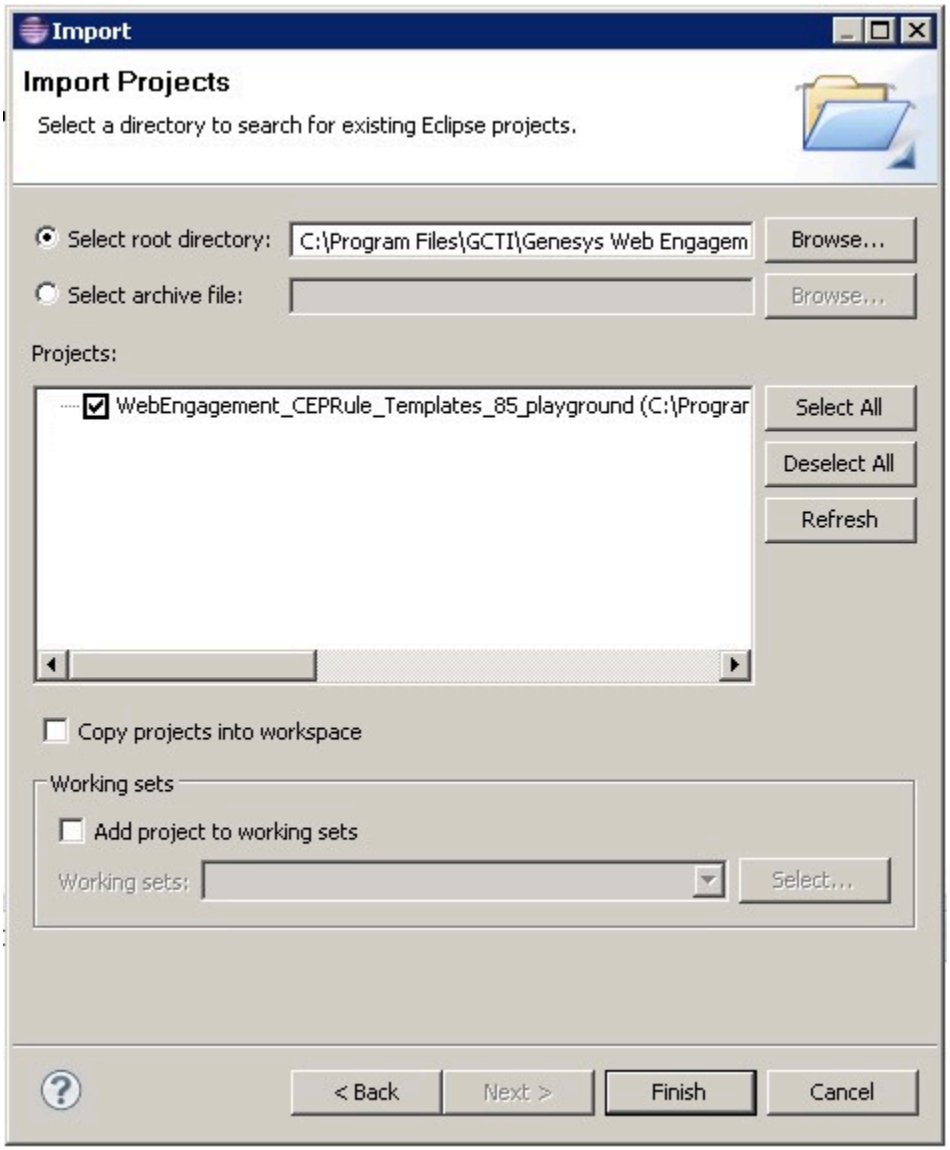
Prerequisites

- The Genesys Rules Development Tool is installed, configured, and opened in Composer.

Start

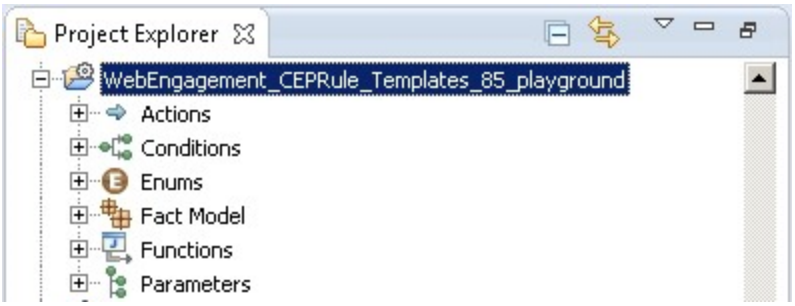
1. Navigate to **Window > Open Perspective > Other > Template Development** to switch to the Template Development perspective of the Genesys Rules Development Tool.
2. Select **File > Import...**.
3. In the **Import** dialog window, navigate to **General > Existing Projects into Workspace**. Click **Next**.
4. Select **Select Root Directory:**, then click **Browse**.
5. Import your project from ***Web Engagement installation directory*\apps*application name*\resources_composer-projects\WebEngagement_CEPRule_Templates**:

- Browse to the **\apps*application name*\resources_composer-projects** folder in the Genesys Web Engagement installation directory and select a project.
- Click **OK**. **WebEngagement_CEPRule_Templates*application name*** is added to the **Projects** list.
- Select the **WebEngagement_CEPRule_Templates*application name*** project.
- Warning: Do **not** enable the option **Copy projects into workspace**.



Import the default templates by clicking **Finish**.

- Click **Finish** to import the project. **WebEngagement_CEPRule_Templates*application name*** is added to the **Project Explorer**.



WebEngagement_CEPRule_Templates*application name* is added to the Project Explorer.

End

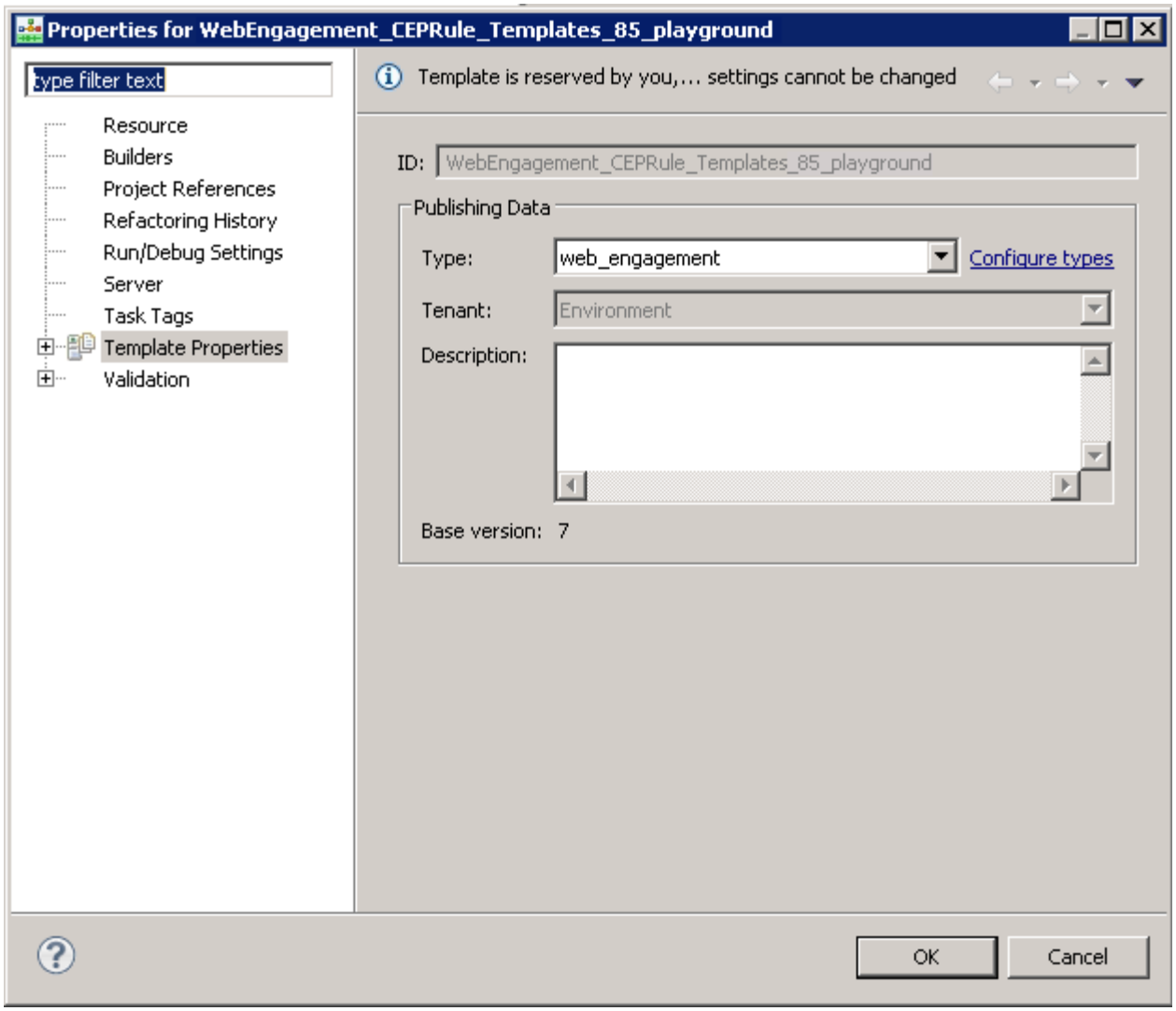
Configuring the CEP Rule Templates

Prerequisites

- The **Web Engagement Categories** business attribute is defined in Genesys Administrator.

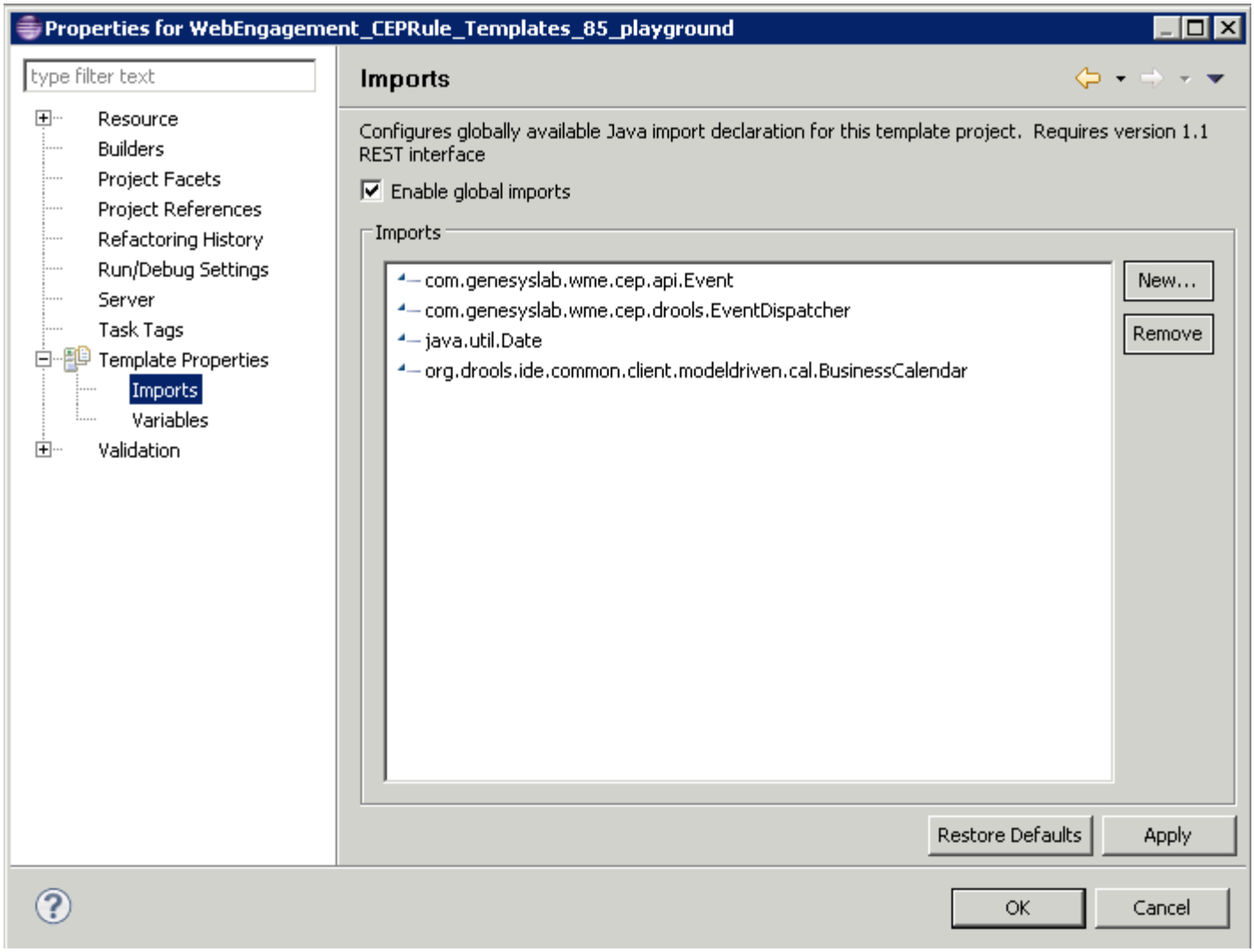
Start

1. In the GRDT **Project Explorer**, right-click on the **WebEngagement_CEPRule_Templates*application name*** project. Click **Properties**.
2. In the **Properties** dialog window, navigate to **Template Properties**. In **Publishing Data**, set **Type** to `web_engagement`.



Set the **type** to `web_engagement`.

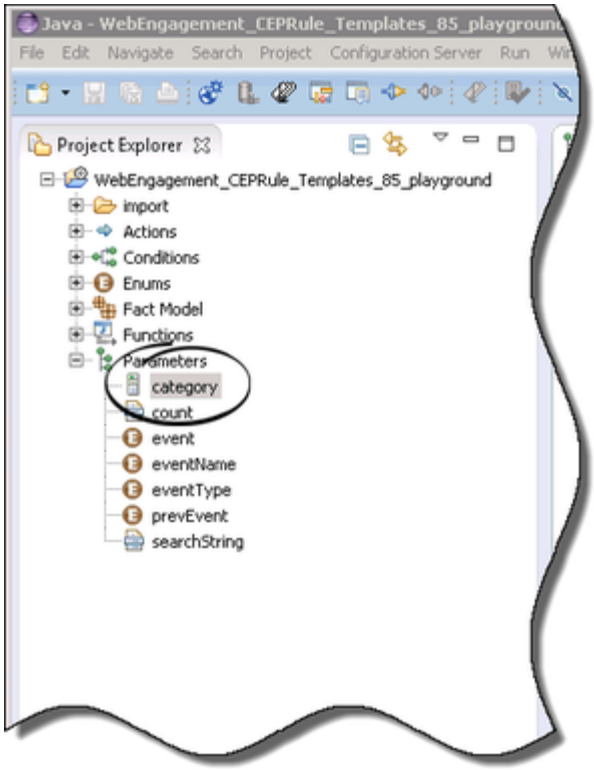
3. Navigate to **Template Properties > Imports**. The **Imports** panel opens.
4. Select the **Enable global imports** option.



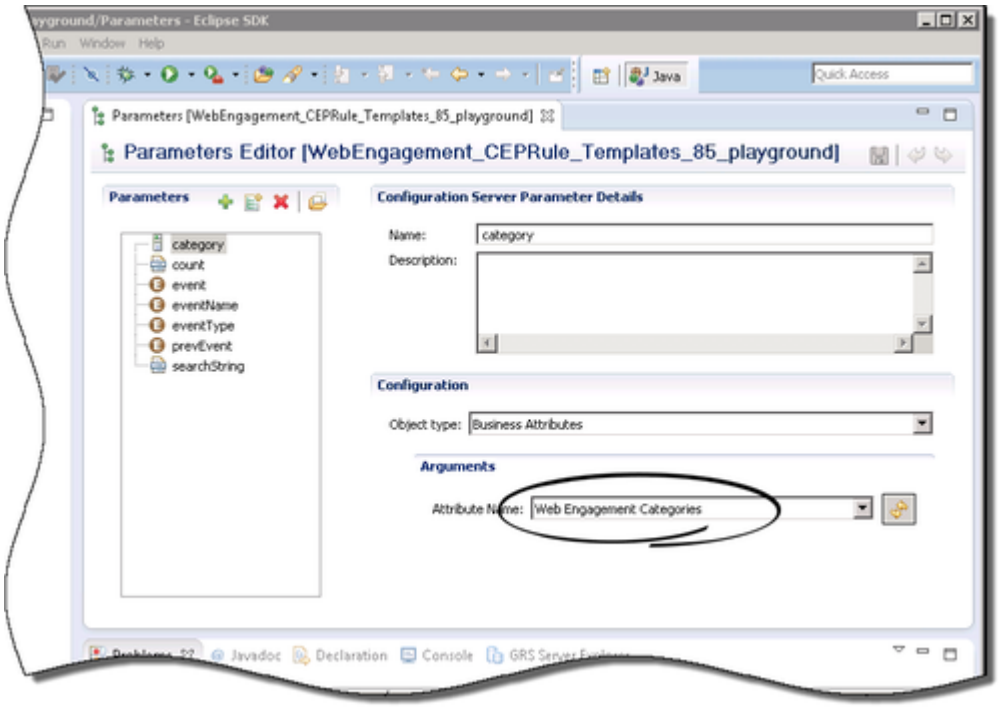
Enabling global imports.

Note: The **com.genesyslab.wme.cep.api.Event** and **com.genesyslab.wme.cep.drools.EventDispatcher** packages must be present.

5. Click **OK**.
6. In the **Project Explorer**, navigate to **WebEngagement_CEPRule_Templates*application name*** > **Parameters** > **category**.



7. In the **Parameters Editor** panel, set **Attribute Name** to `Web Engagement Categories`.



8. Click **Save**.

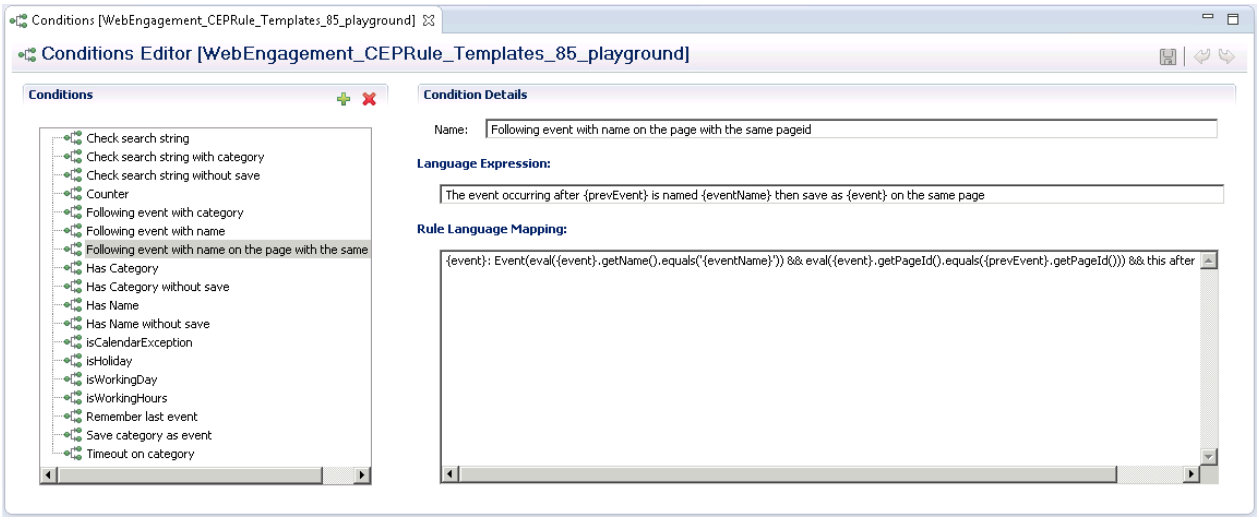
End

Customizing the CEP Rule Templates (Optional)

Start

1. Open the CEP rule template project with GRDT and navigate to the Conditions item.
2. Expand Conditions to open the Conditions editor.
3. In the Conditions tab, click **+**. The **Add Condition** window opens.

4. Enter a name and click **OK**. The condition is added and selected in the condition list; the condition detail panel opens.
5. Insert the Language Expressions and Rule Language Mapping:



6. Click Save Now when the rule template is published, the rule will be available in GRAT:

End

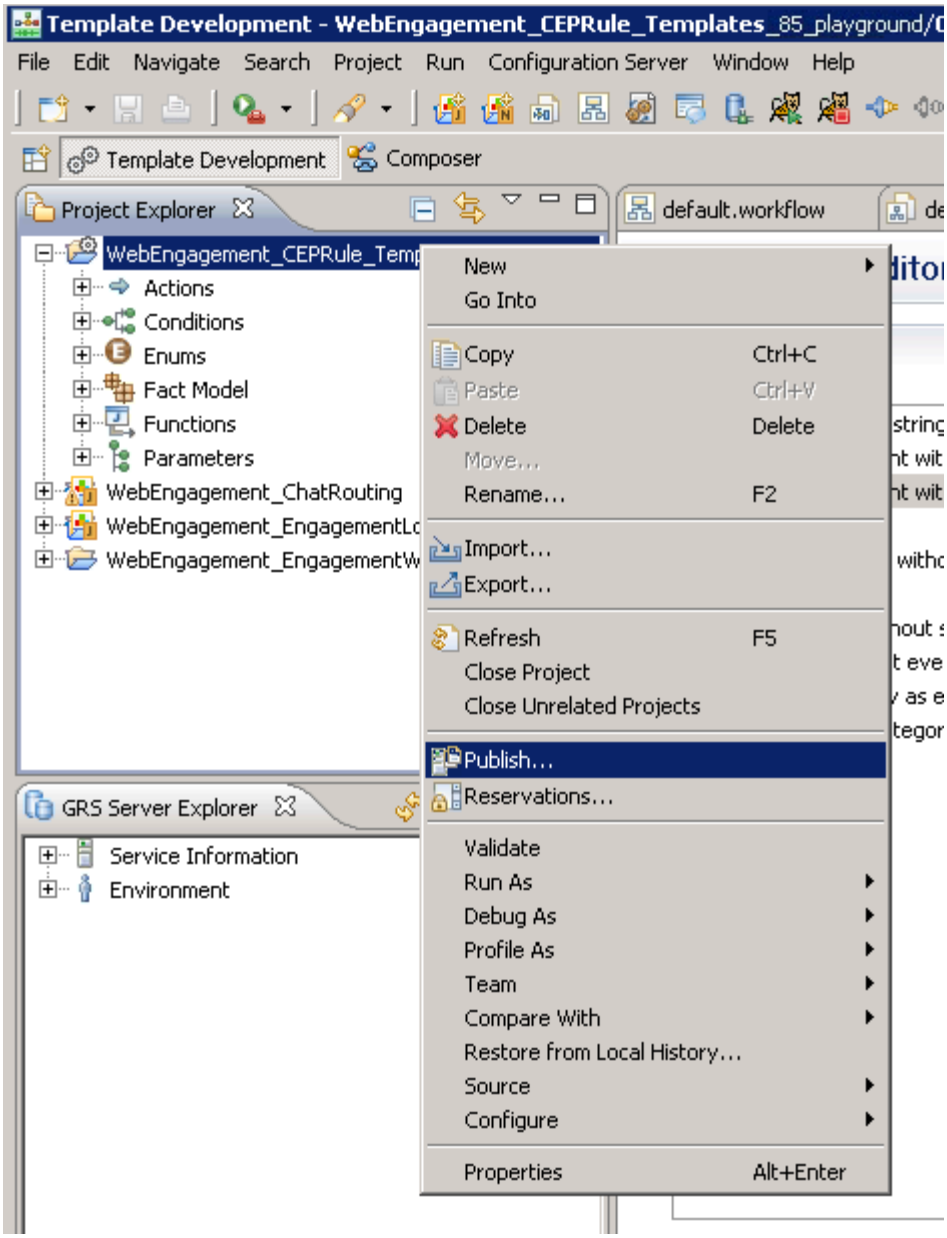
Publishing the CEP Rule Templates in the Rules Repository

Prerequisites

- Your user has the correct permissions to manage rules in GRAT, as detailed in the Genesys Rules System Deployment Guide.
- You configured GRDT to enable a connection to Configuration Server and Rules Repository Server.

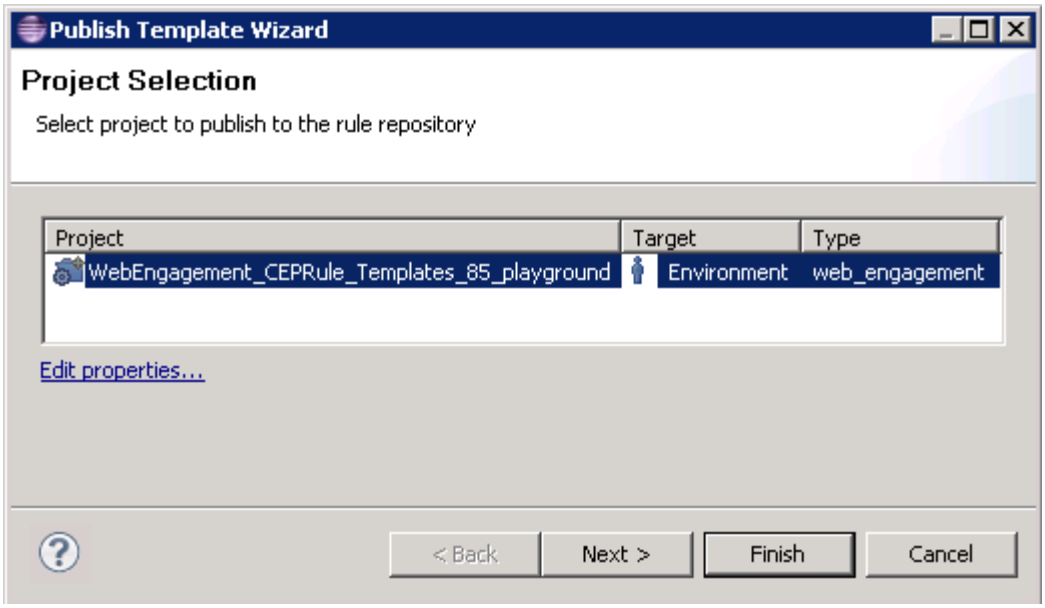
Start

1. In **Project Explorer**, right click **WebEngagement_CEPRule_Templates*application name***.
2. Select **Publish**. The **Publish Template Wizard** opens.



The Publish Template Wizard.

3. Select **WebEngagement_CEPRule_Templates*application name***.



Select **WebEngagement_CEPRule_Templates*application name***.

4. Click **Finish**.

End

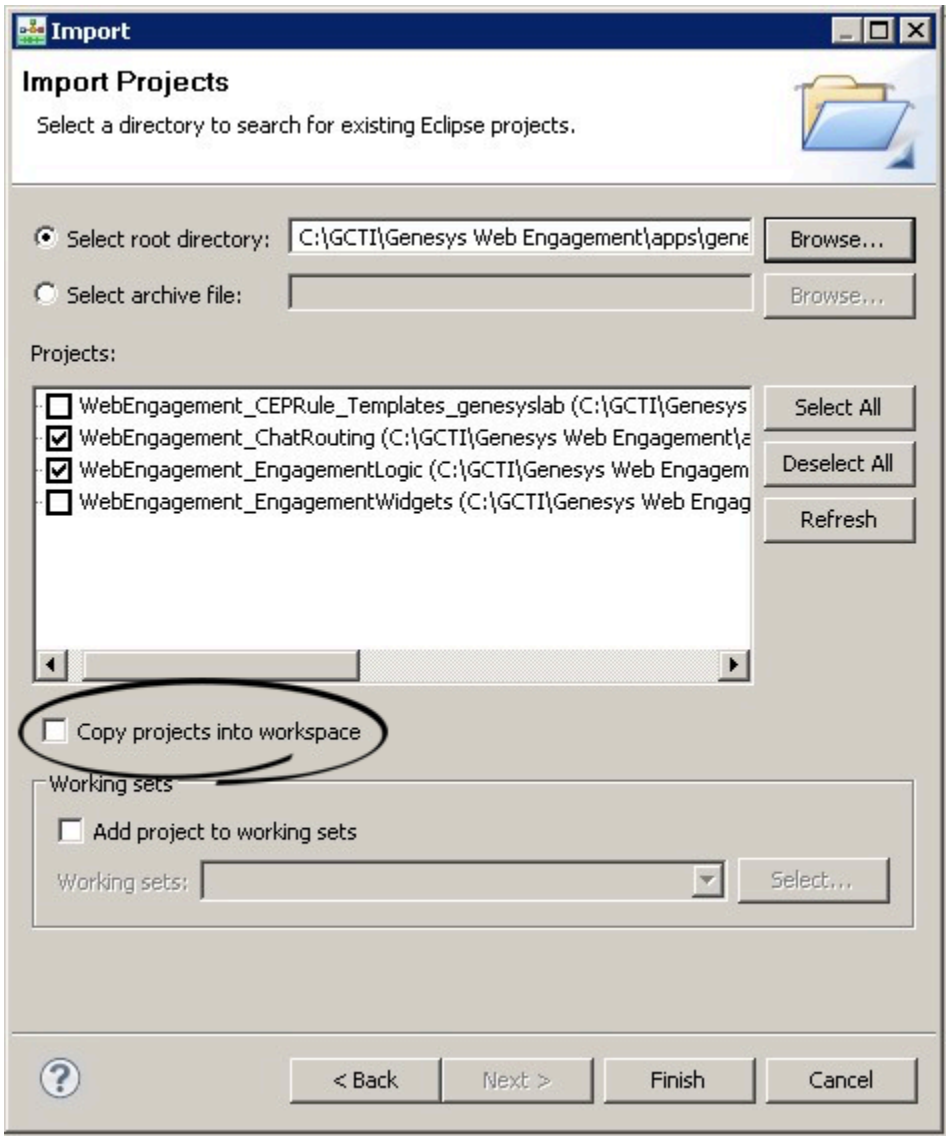
Next Steps

- You can continue customizing your application:
  - Customizing the SCXML Strategies
  - Customizing the Browser Tier Widgets
- You can deploy your application.

Customizing the SCXML Strategies

When you create your application, Genesys Web Engagement also creates default chat routing and engagement logic 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 custom widget, for example, an advertisement). You can modify these strategies by importing them into Composer.

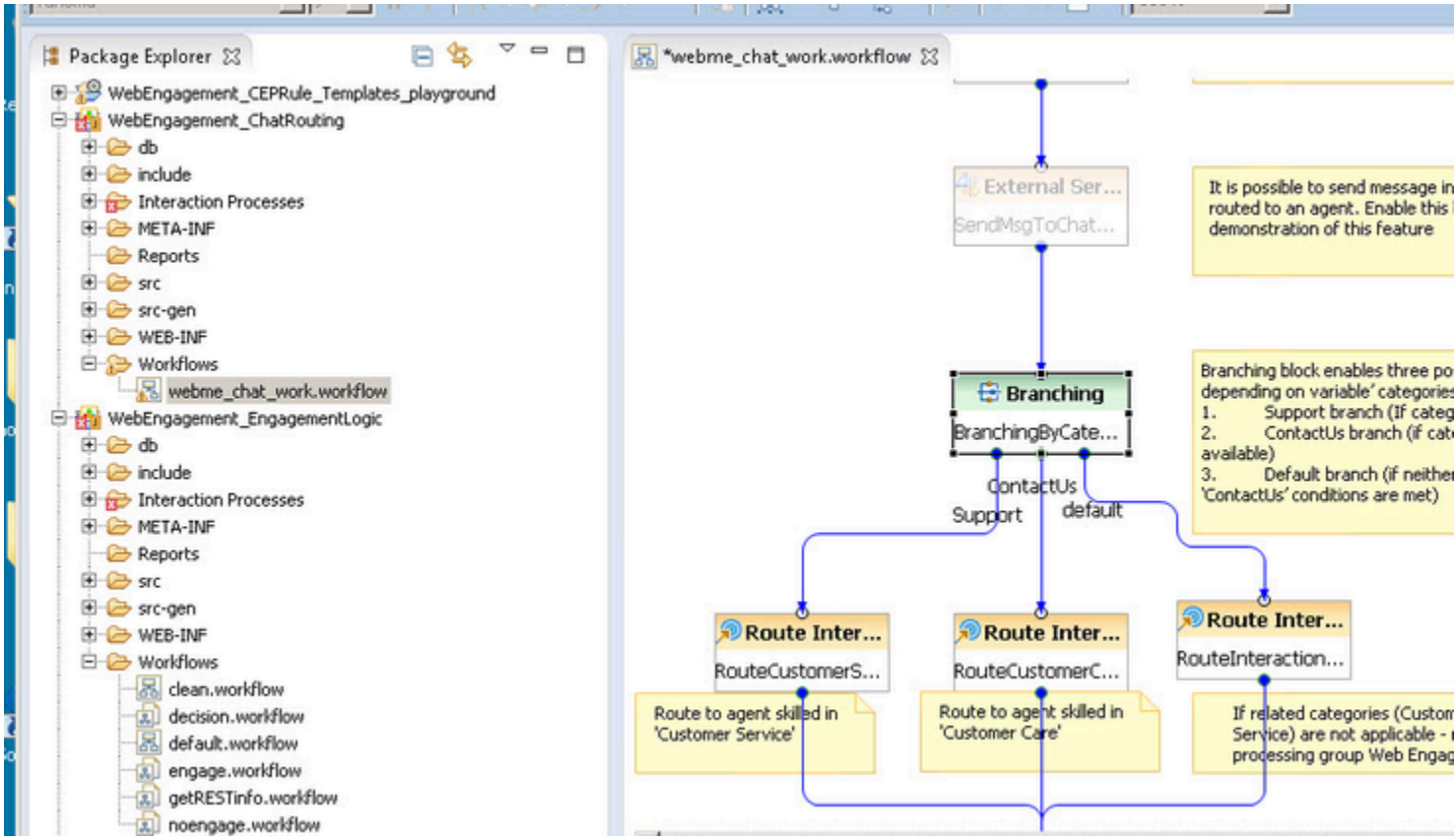
Warning: When importing routing strategies, you must not enable the **Copy projects into workspace** option.



Routing Strategy Import Dialog

Warning: When you modify your routing strategies, you can update workflows and processes. You can also compile the strategies. But you must not use Composer's Publish functionality, which is incompatible with Web Engagement.

The following shows the Chat Routing workflow, where interactions are routed to agents with "Customer Service" or "Customer Care" skills:



A Chat Routing workflow example.

When you alter the strategies, you must save your changes, generate the code, redeploy, and restart your Genesys Web Engagement application to apply those changes.

You can customize the routing strategies to help meet your specific business needs:

- Customizing the Engagement Strategy
- Customizing the Chat Routing Strategy

Customizing the Engagement Strategy

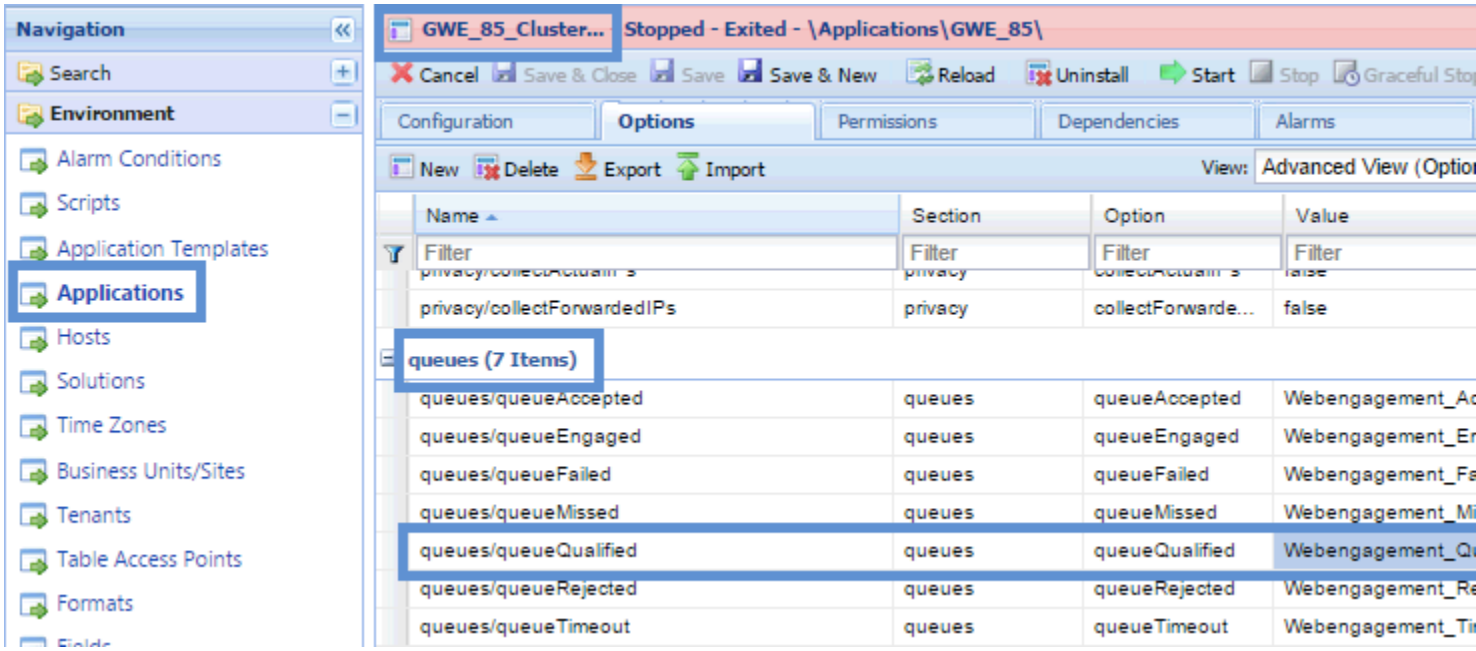
When you create your Web Engagement application, Genesys Web Engagement also creates default Engagement Logic and Chat Routing SCXML strategies in the **\apps*application_name*\resources_composer-projects** folder. Orchestration Server (ORS) uses these strategies to decide whether and when to make a 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.



The Interaction Queue

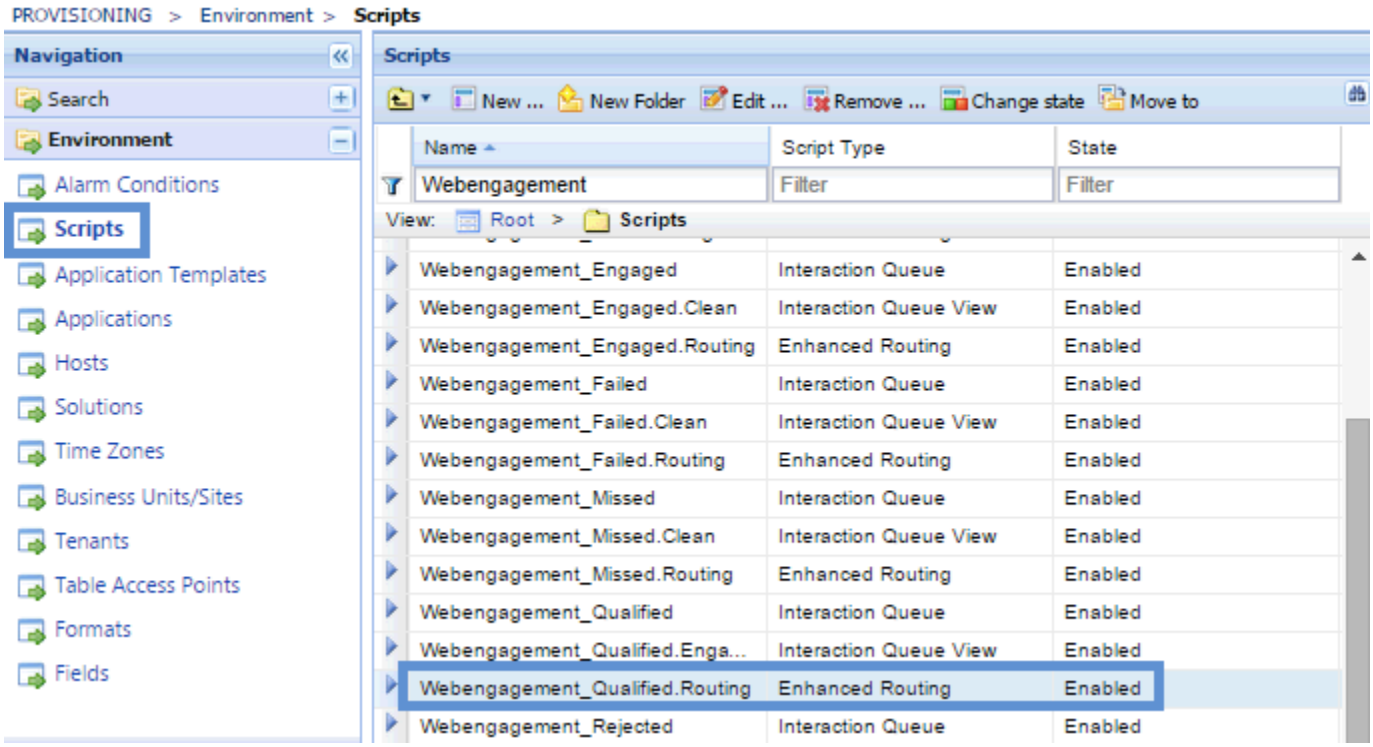
Passing Parameters into the Engagement Logic Strategy

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.

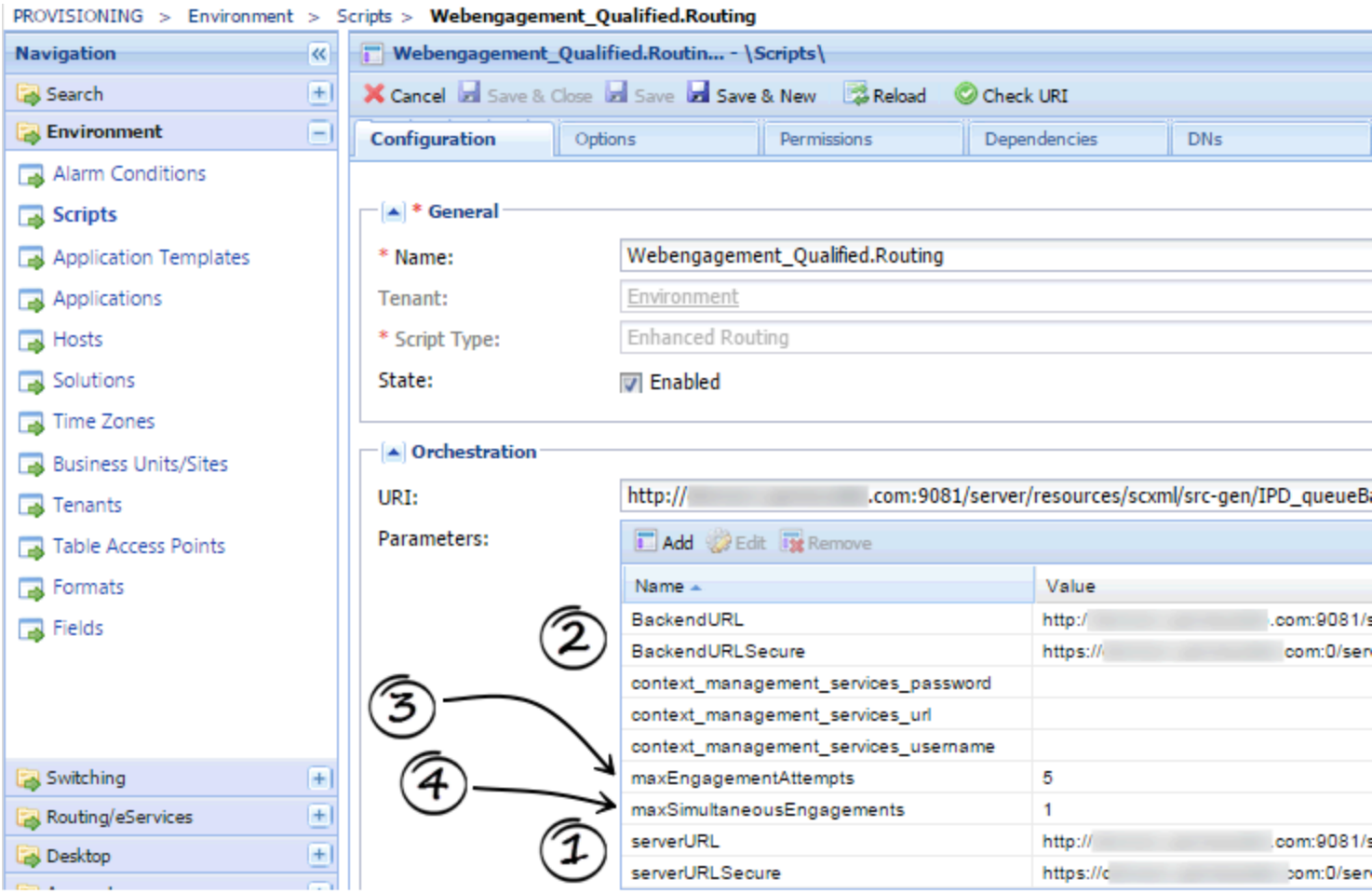
1. The address where the SCXML strategy is located. **Note:** The default Engagement Logic and Chat Routing strategies are located as resources under the Web Engagement Server. Provisioning automatically specifies this address in the related Configuration Server objects when GWE is installed. Since you 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.



The Webengagement_Qualified.Routing Script Object

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



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:

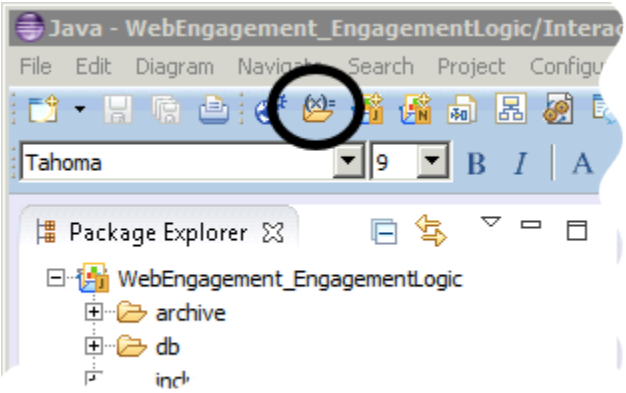
- Event traffic uses a non-secure server (as specified by the **serverURL** parameter), but you need a secure connection for your chat traffic (in which case **mediaServerURL** will specify an HTTPS endpoint)
- Event traffic is processed on one port, but chat traffic needs to be processed on a second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

The Engagement Logic strategy has two interaction processes:

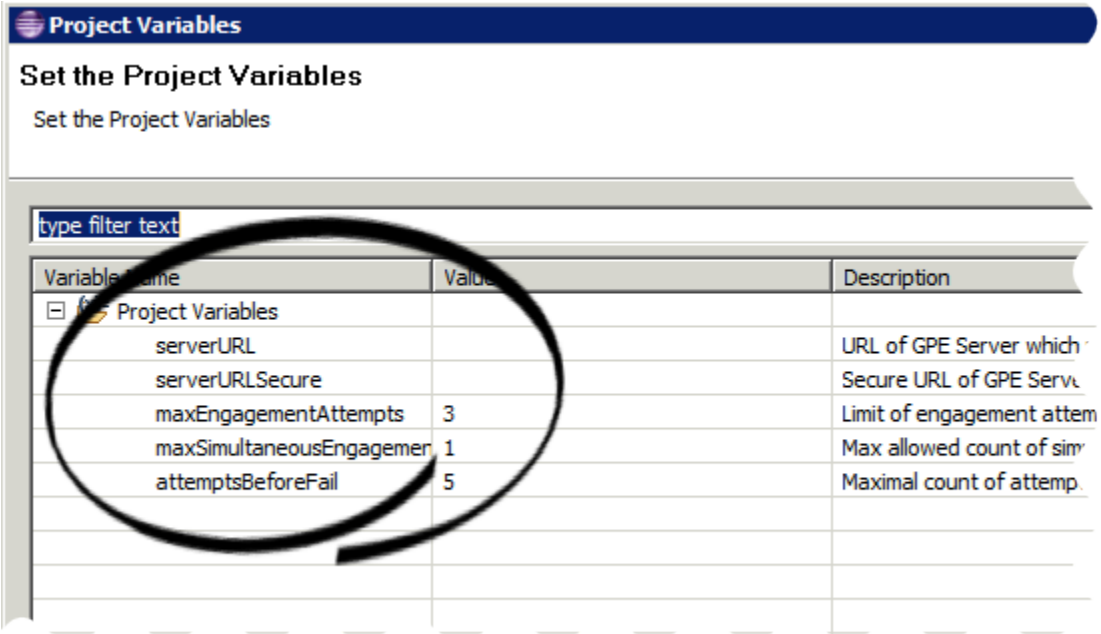
- **clean.ixnprocess** — This process is explained in Cleaning Interaction Process
- **queueBased.ixnprocess** — This process features the major logic for the strategy.

In this section, we will consider the second one.

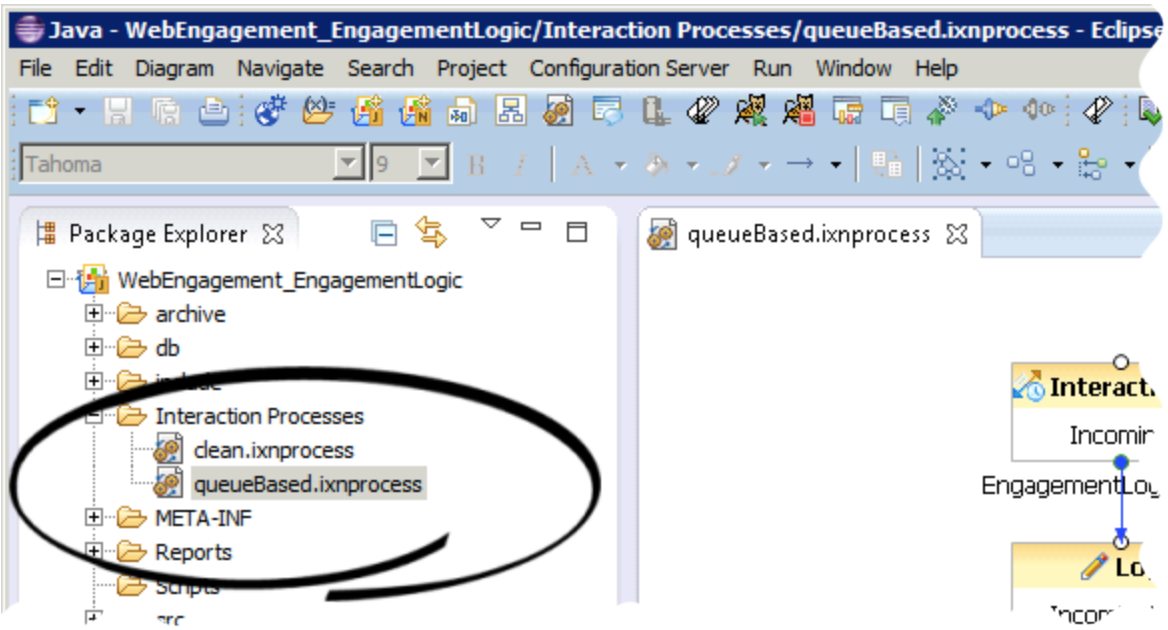
To access the above-mentioned parameters from within Composer, use the Composer **Access Project Variables** button shown in the following image. **Note:** In order to access Project Variables, your current tab in Composer must display Interaction Process (not Workflow).



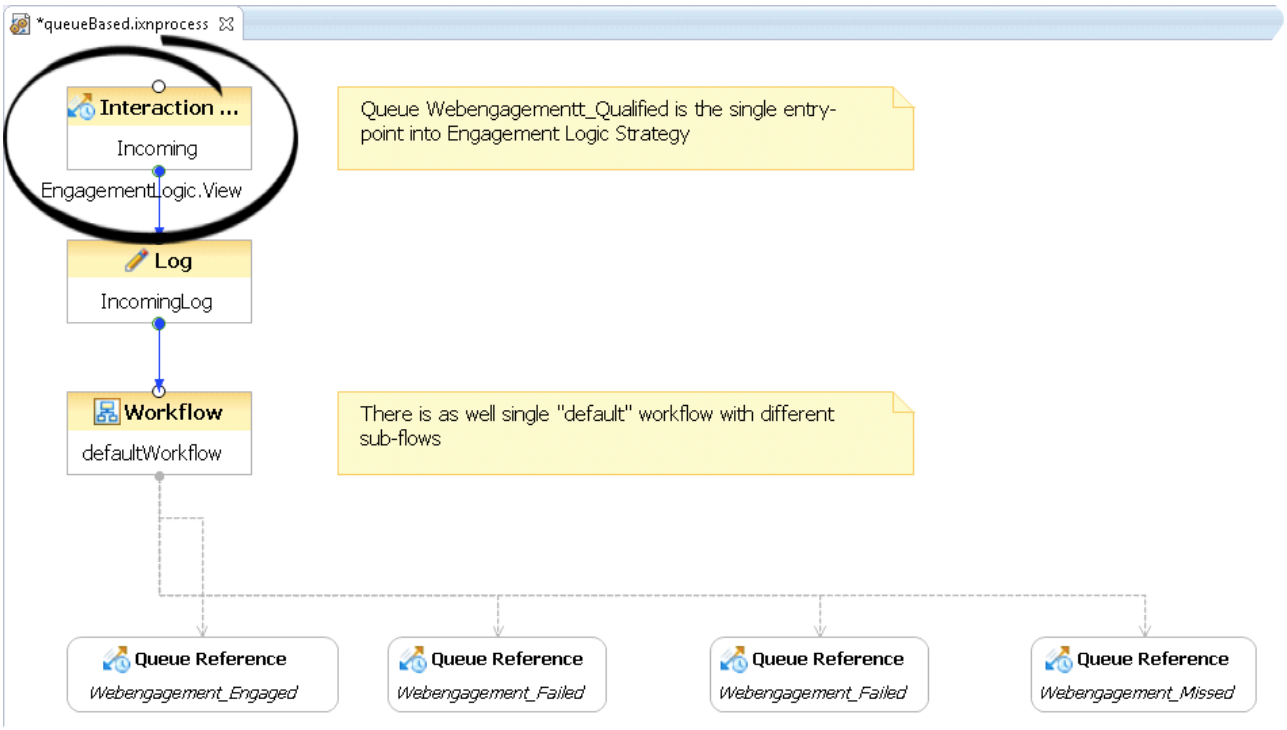
This button opens a window containing the variables we are currently interested in:



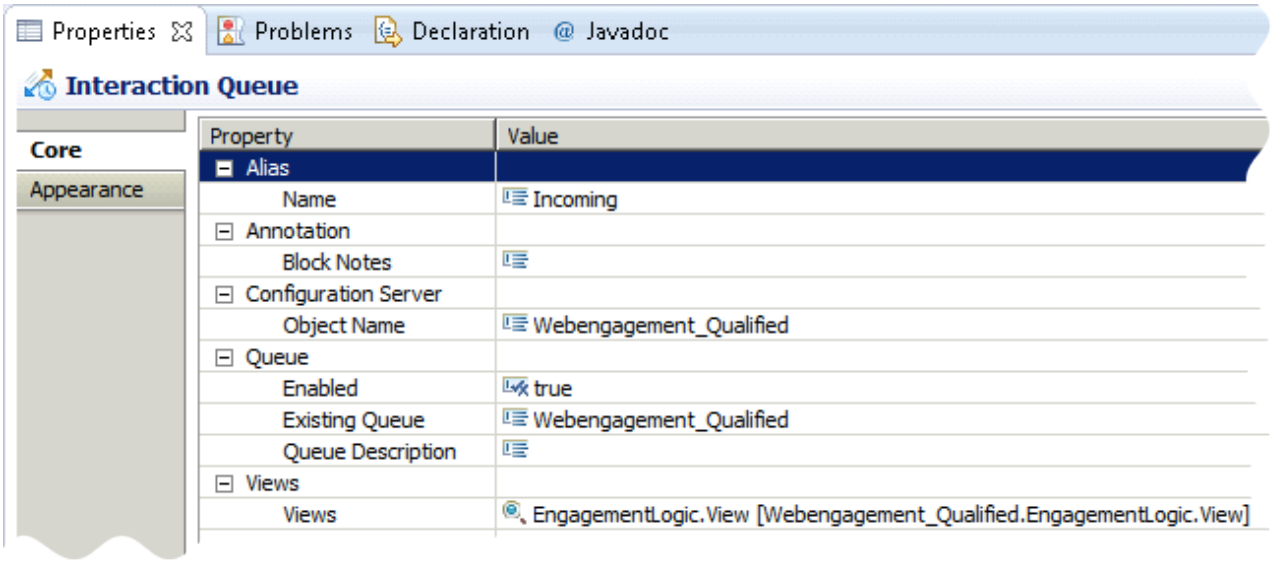
Now let's take a look at **queueBased.ixnprocess**. Select it in the Package Explorer:



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



And its properties are here:



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.

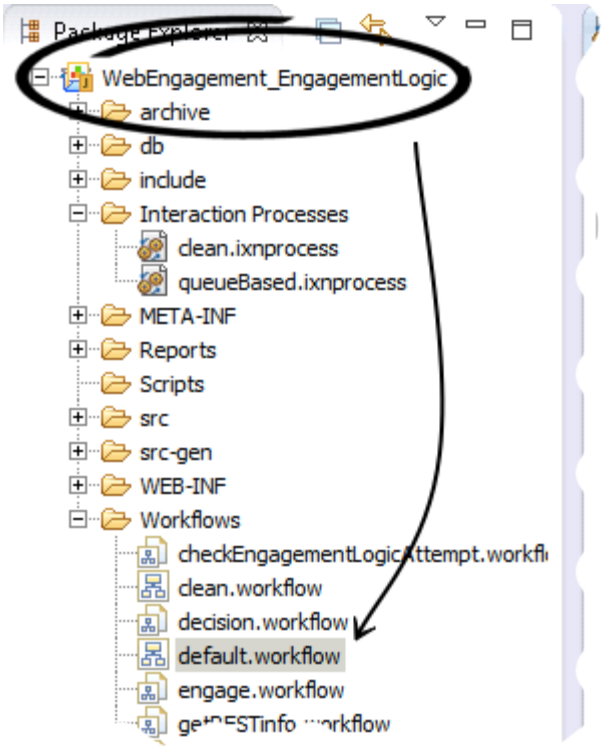
Preventing Interaction Termination into Sub-flows

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.

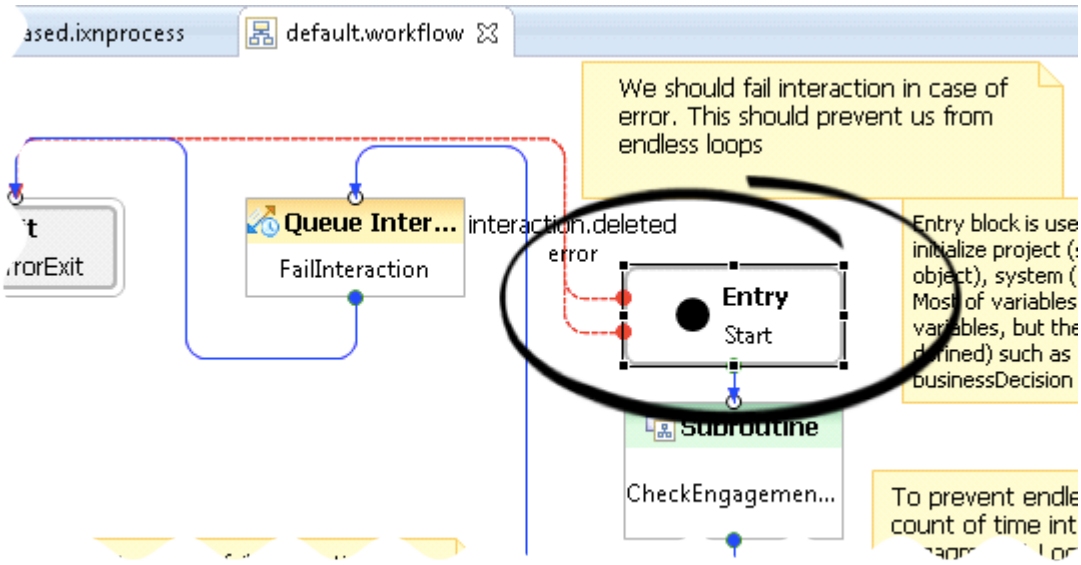
Note: Out-of-the-box Engagement Logic strategies already have the correct specified value (`0`) for the **system.TerminateIxnOnExit** variable.

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

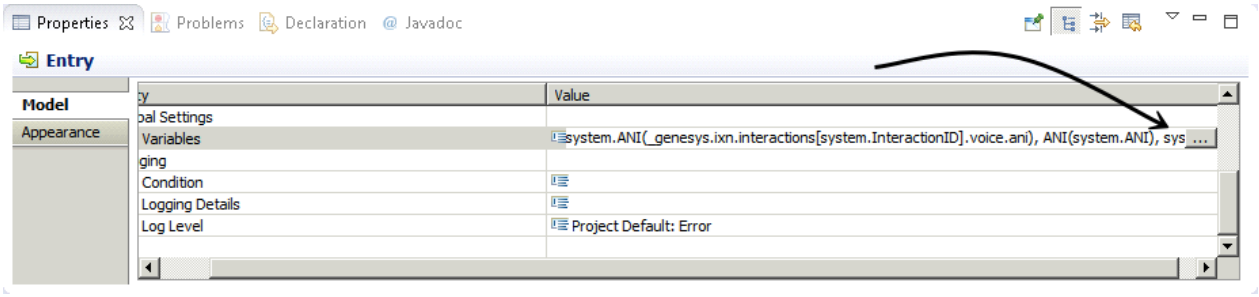
1. Open the workflow diagram in Composer (note that in the images, it is shown as **default.workflow**).



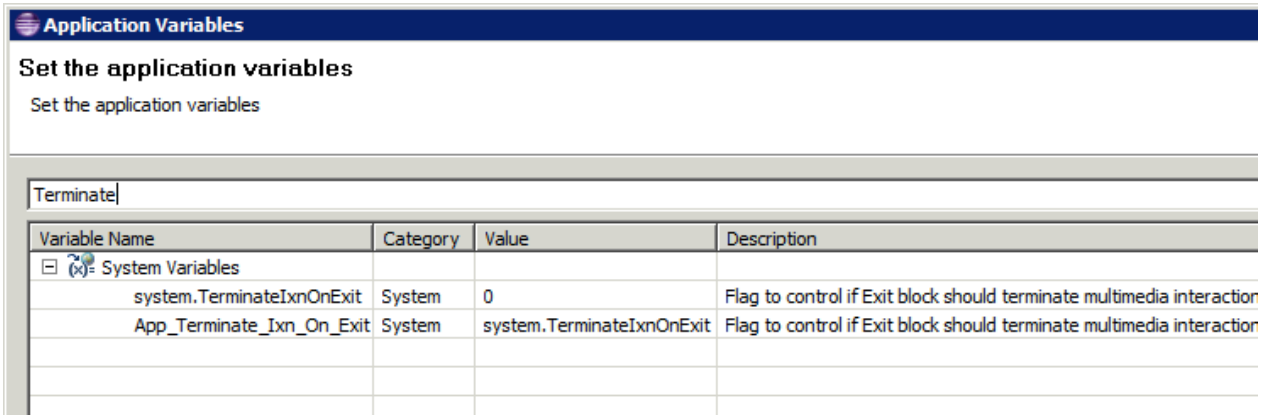
2. Select the **Entry** block.



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



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`.

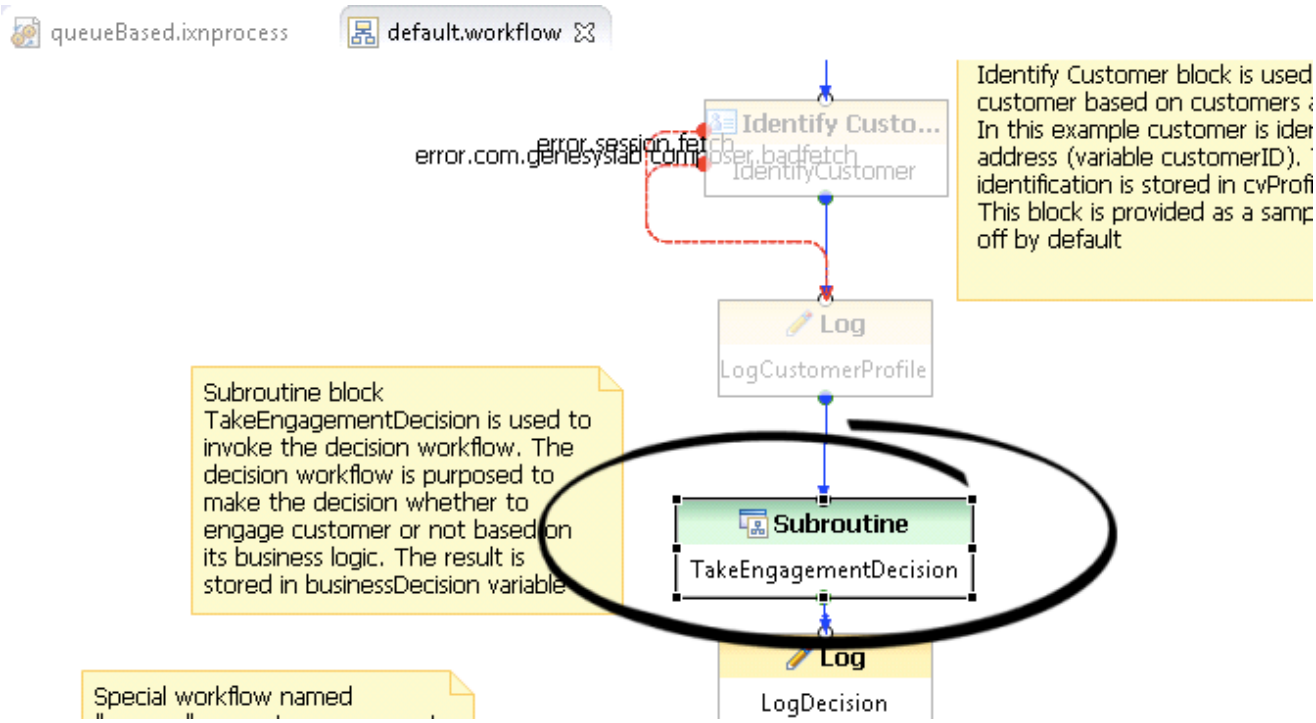


Accessing User Data from the webengagement Interaction and Passing it into Sub-flows

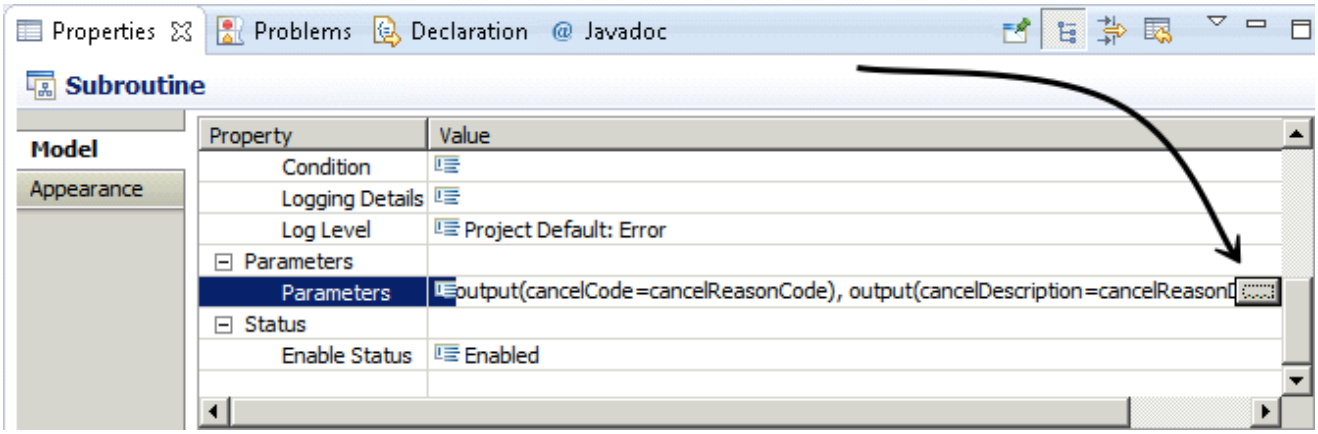
One of the most important features of the Engagement Logic is its ability to access User Data from **webengagement** interactions. This data is populated by the Web Engagement Server and includes, among other things, information provided by a pacing algorithm.

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.

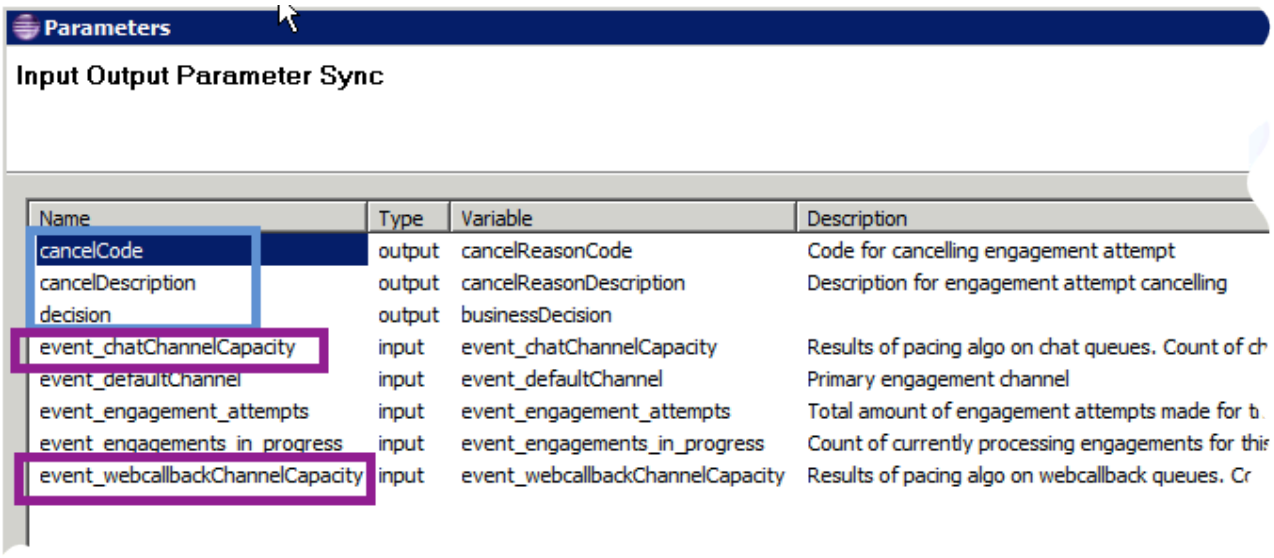
In the following example, we show the **TakeEngagementDecision** subroutine:



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



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**:



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)

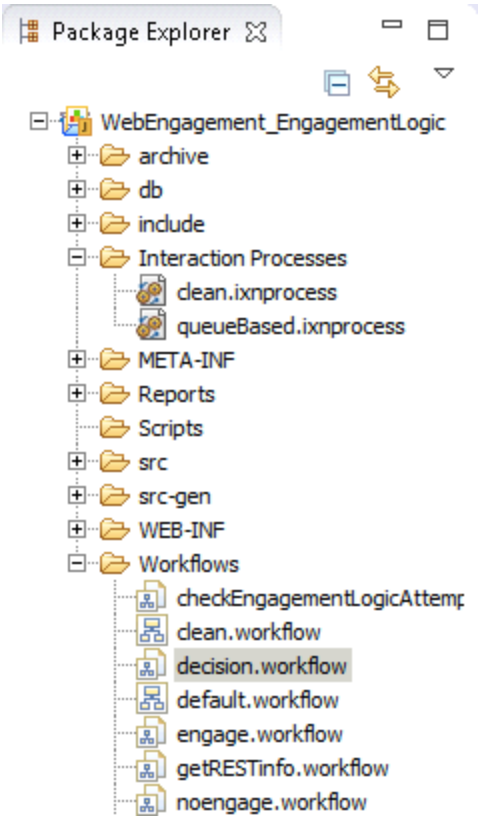
Engagement policy is the other name of decision workflow.

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

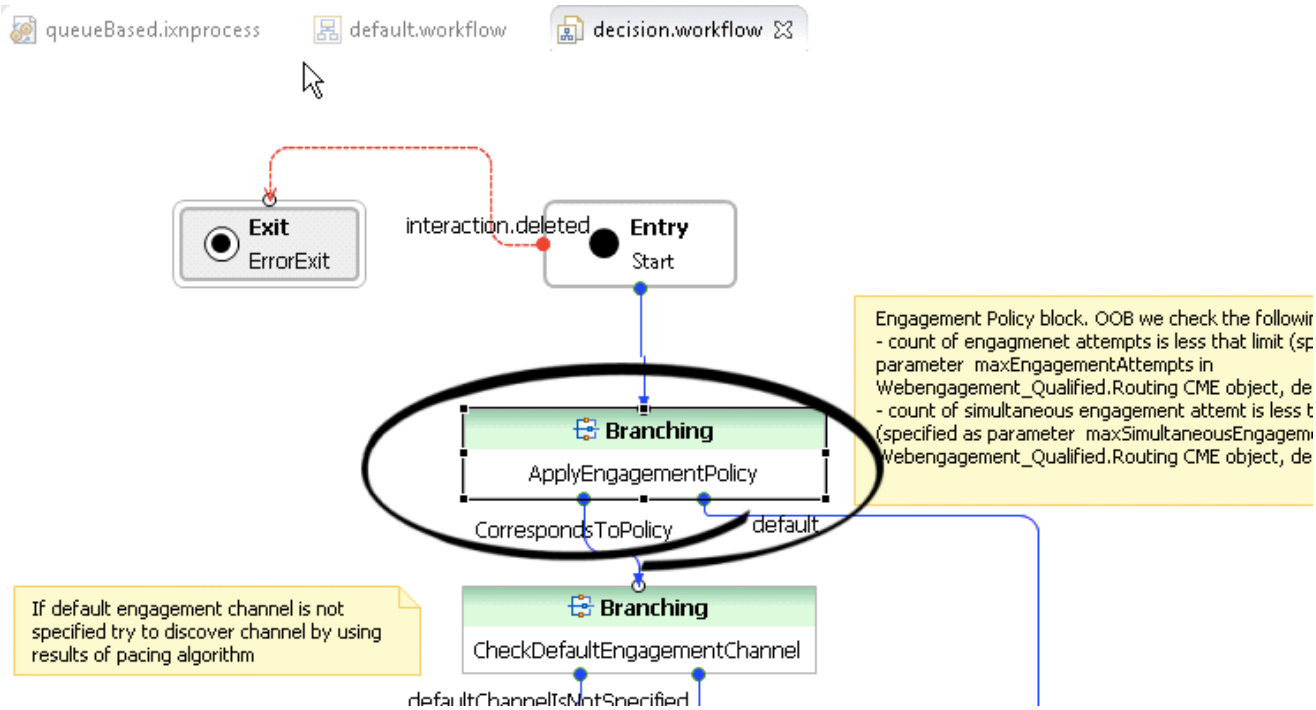
Count of Engagement Attempts

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

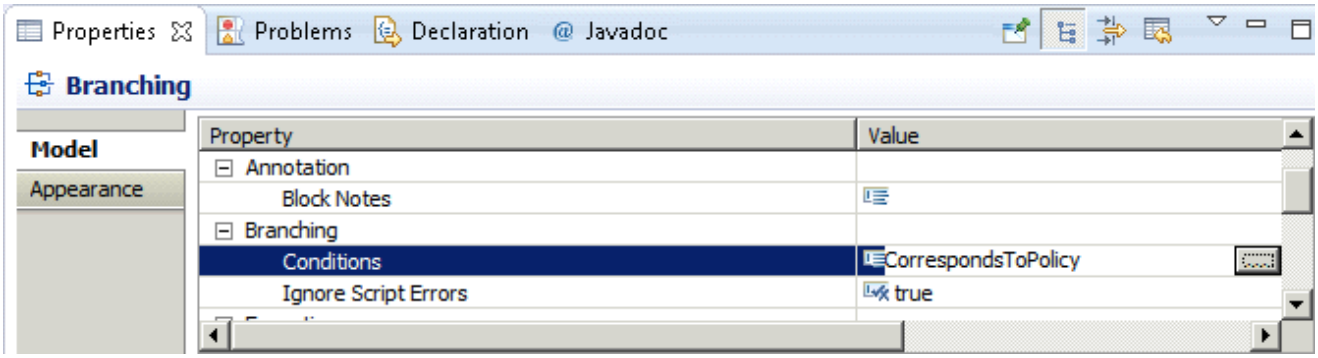
To see where this check is executed open **decision.workflow**:



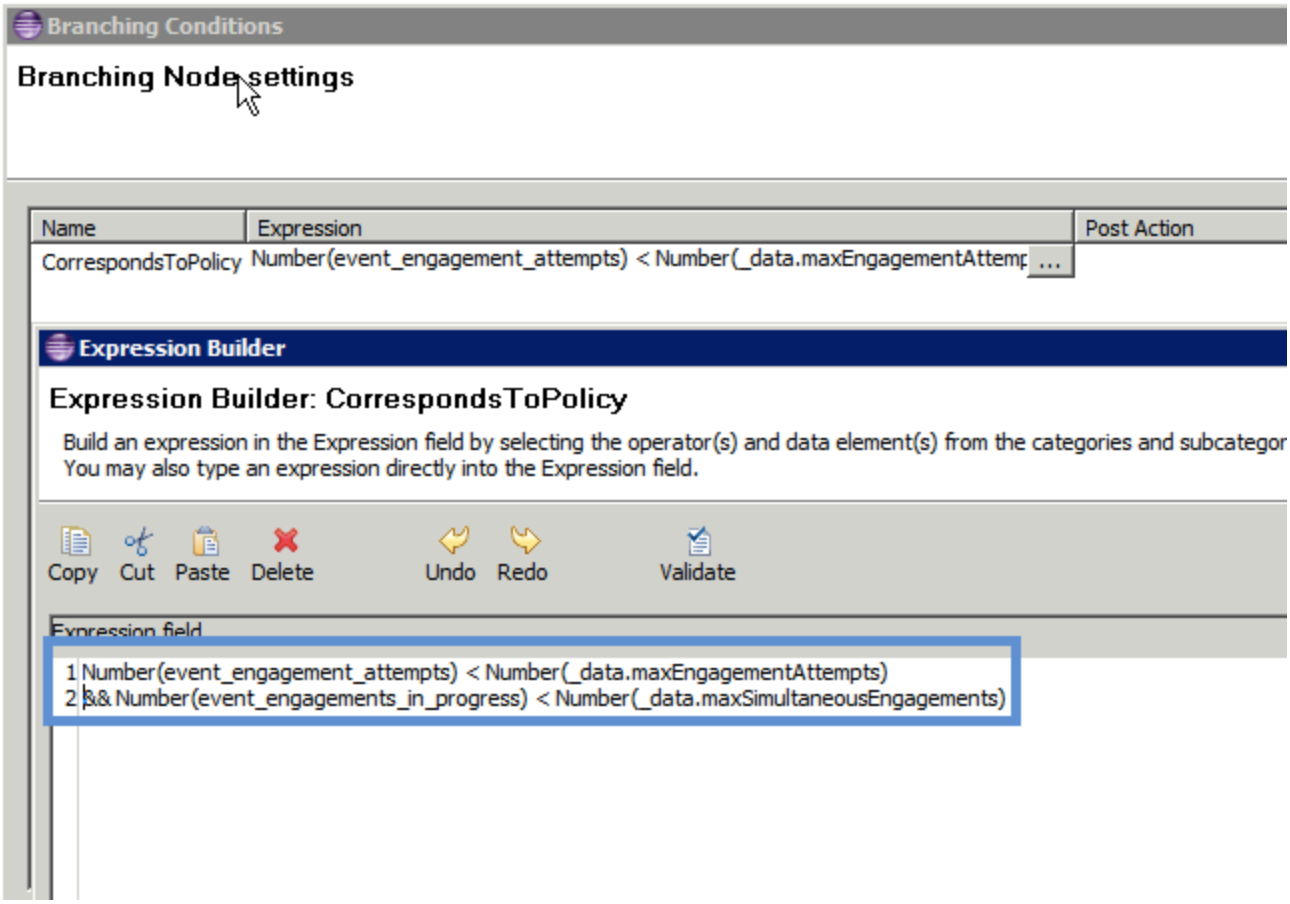
Looking at the workflow, you can select the **ApplyEngagementPolicy** block:



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

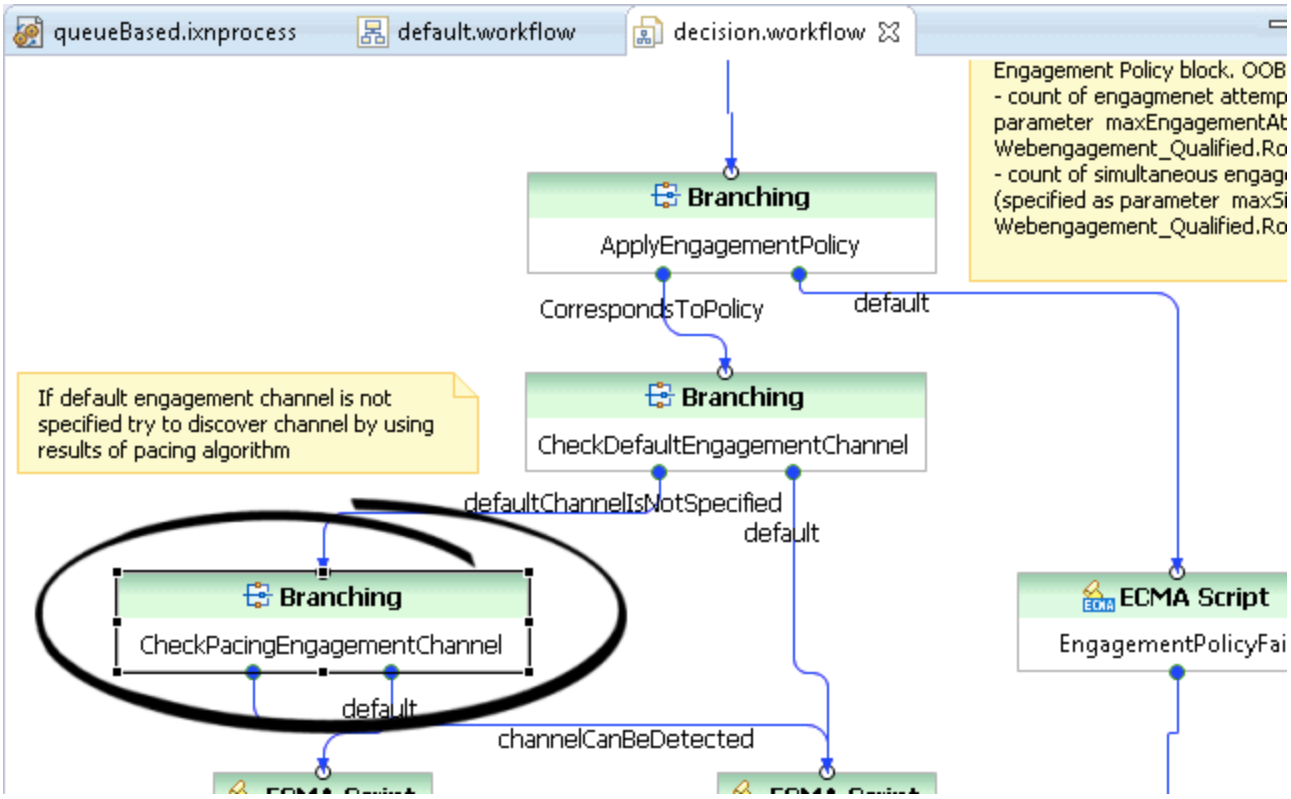


CorrespondsToPolicy is an expression that uses application parameters from the **Webengagement_Qualified.Routing** script object to determine how many engagement attempts should be proposed for a particular visitor. **Note:** Engagement attempts in the current visit that were closed with a timeout disposition 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.

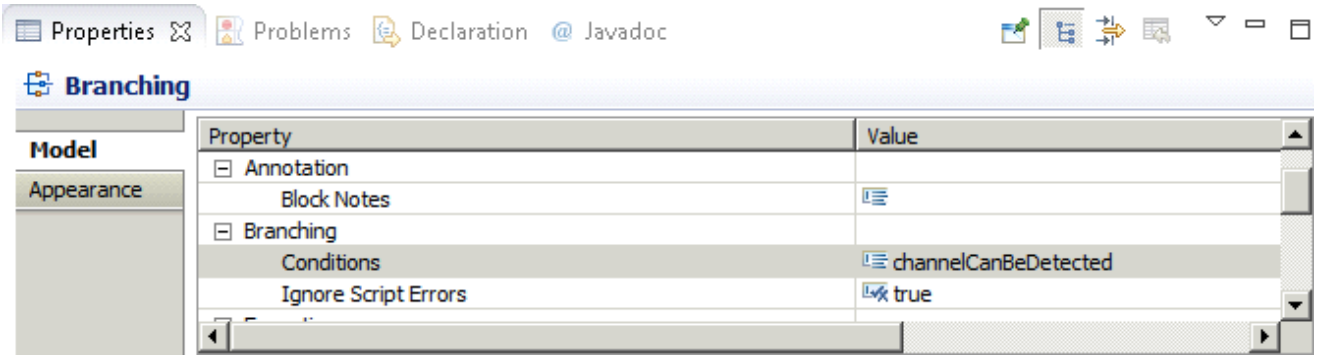


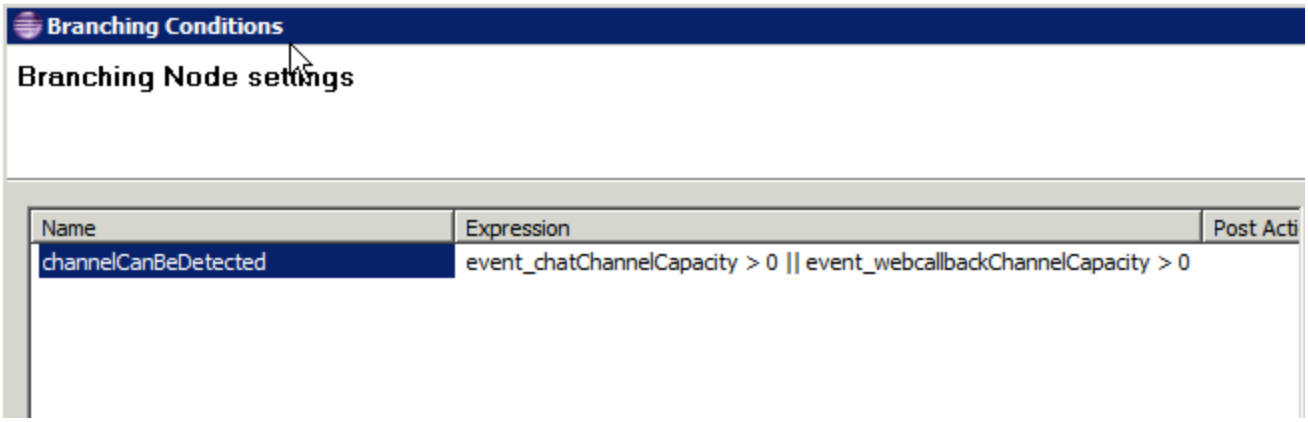
Pacing Information

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



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:





Obtaining Data from the GWE Cassandra Database through REST Requests

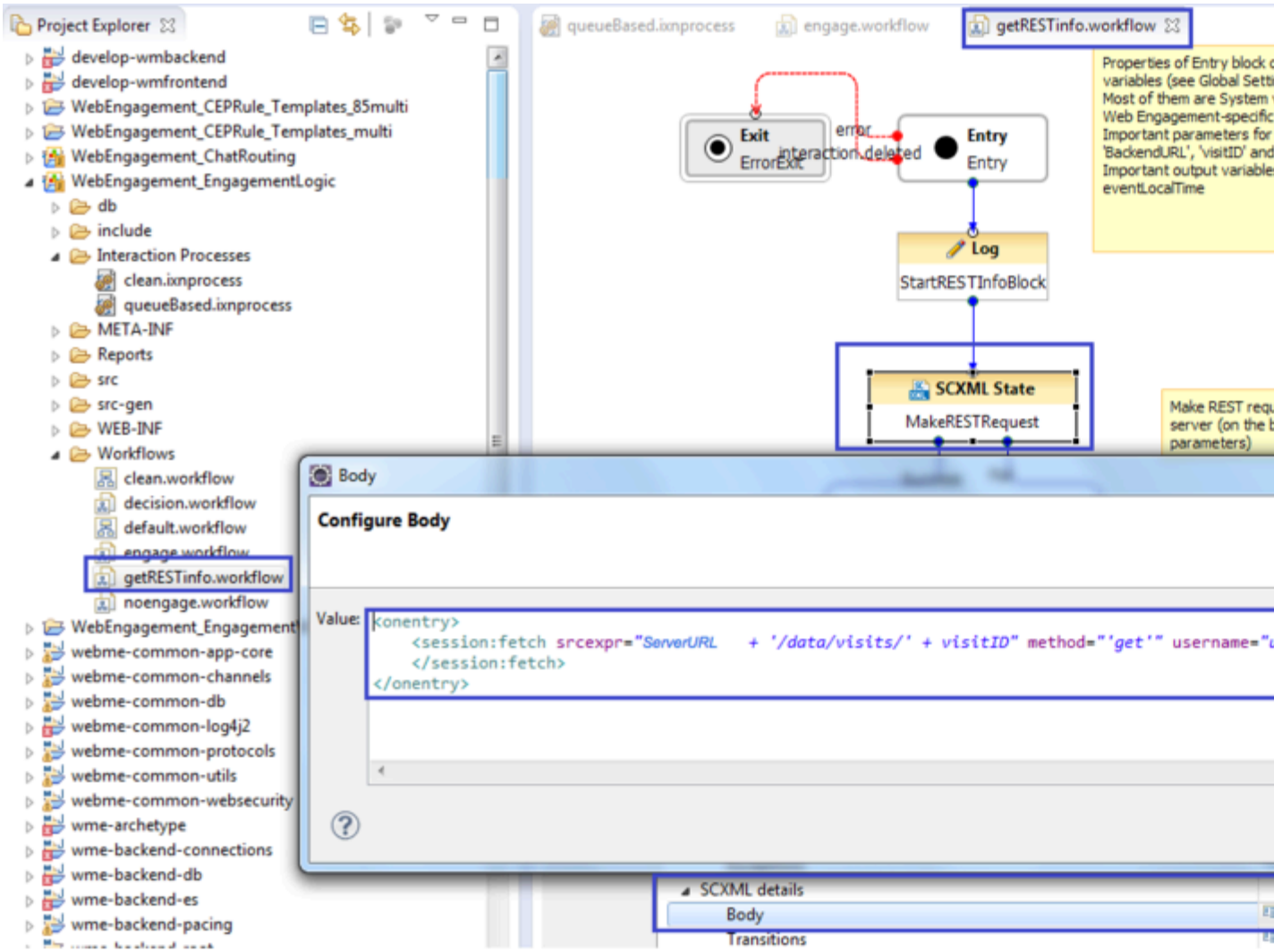
Requesting data from Web Engagement Server through the REST

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.

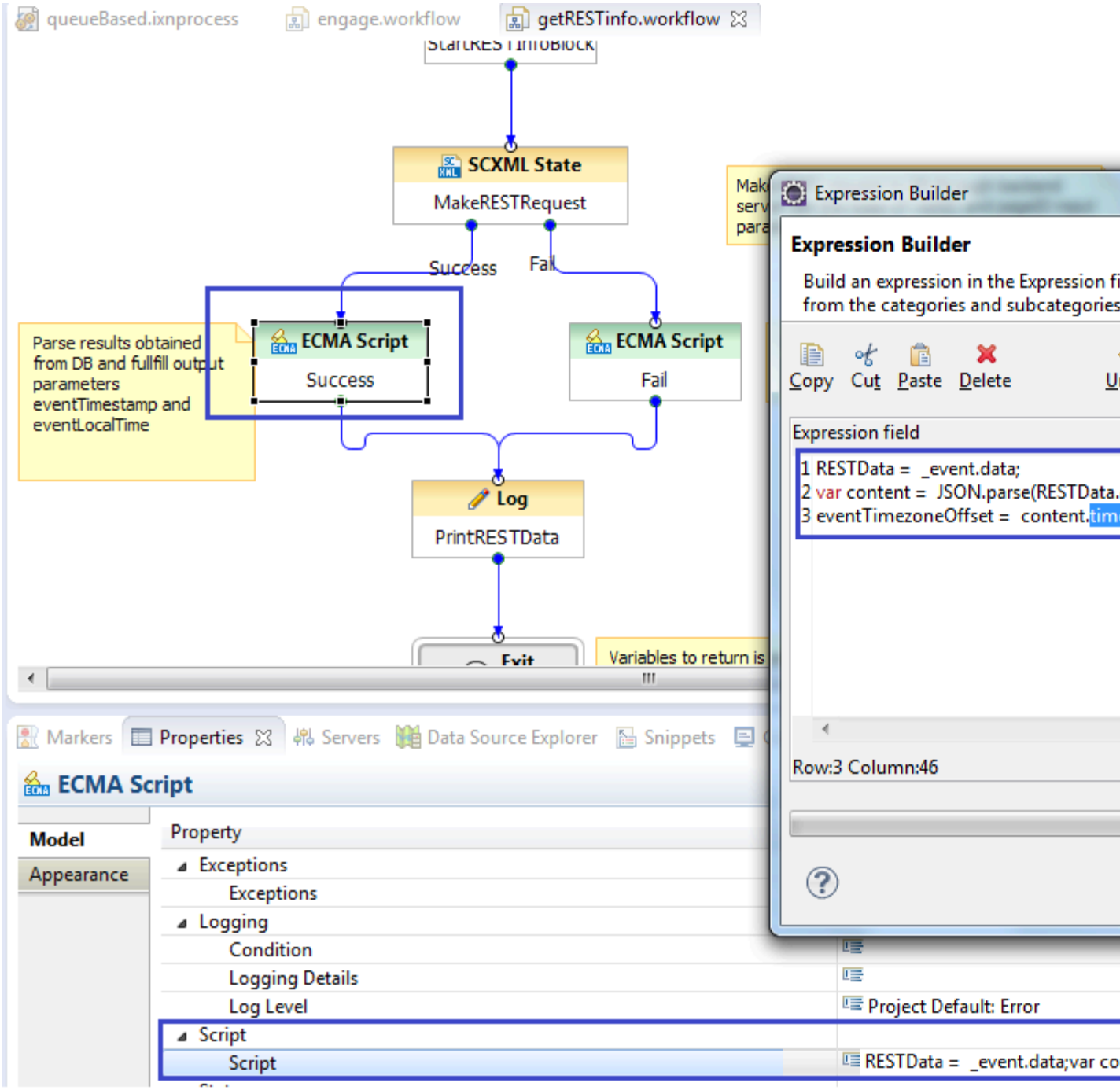
1. Use the **SCXML State** block in order to make the REST request with specified parameters.



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:



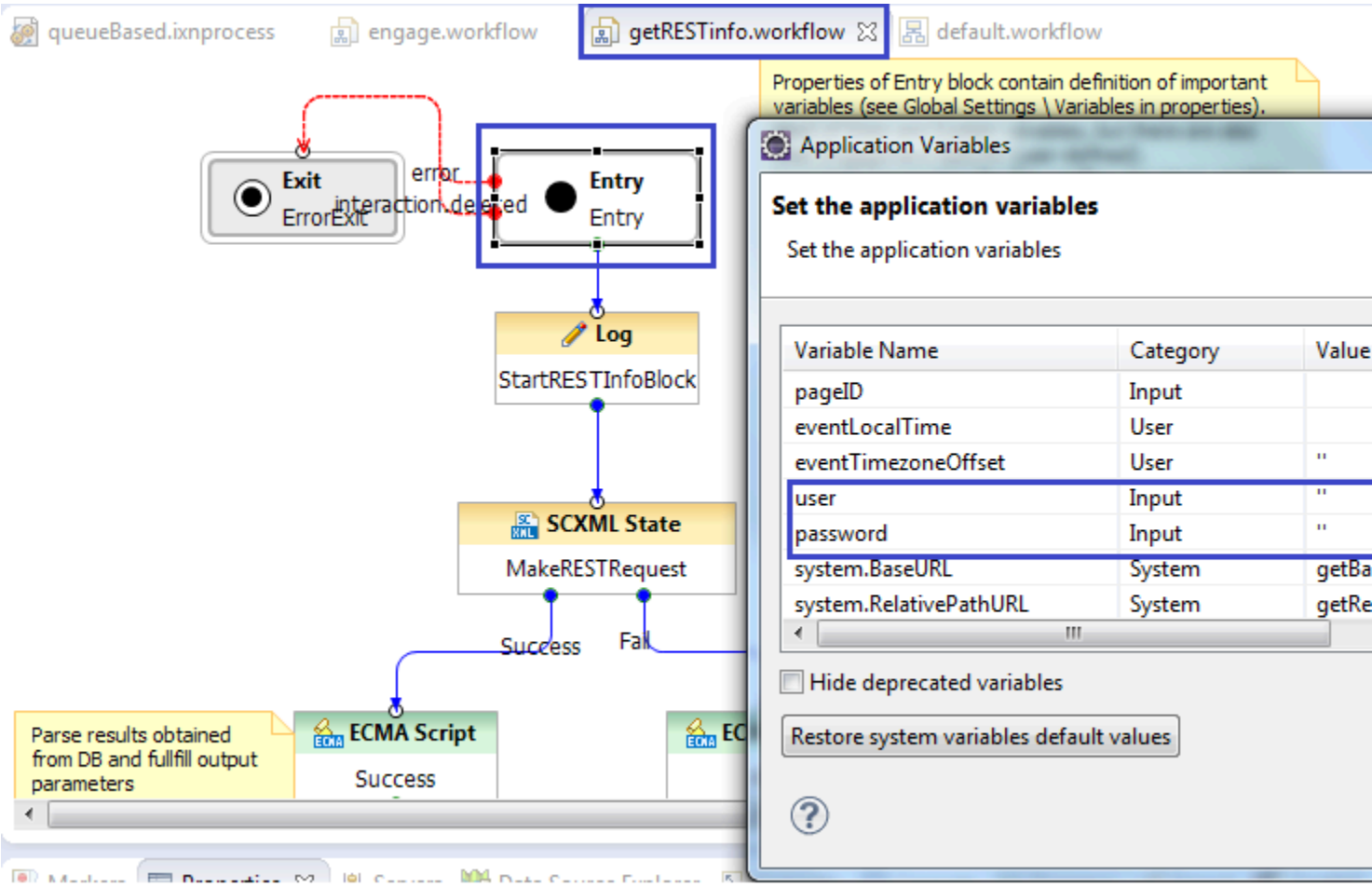
Parse the response to the REST request

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

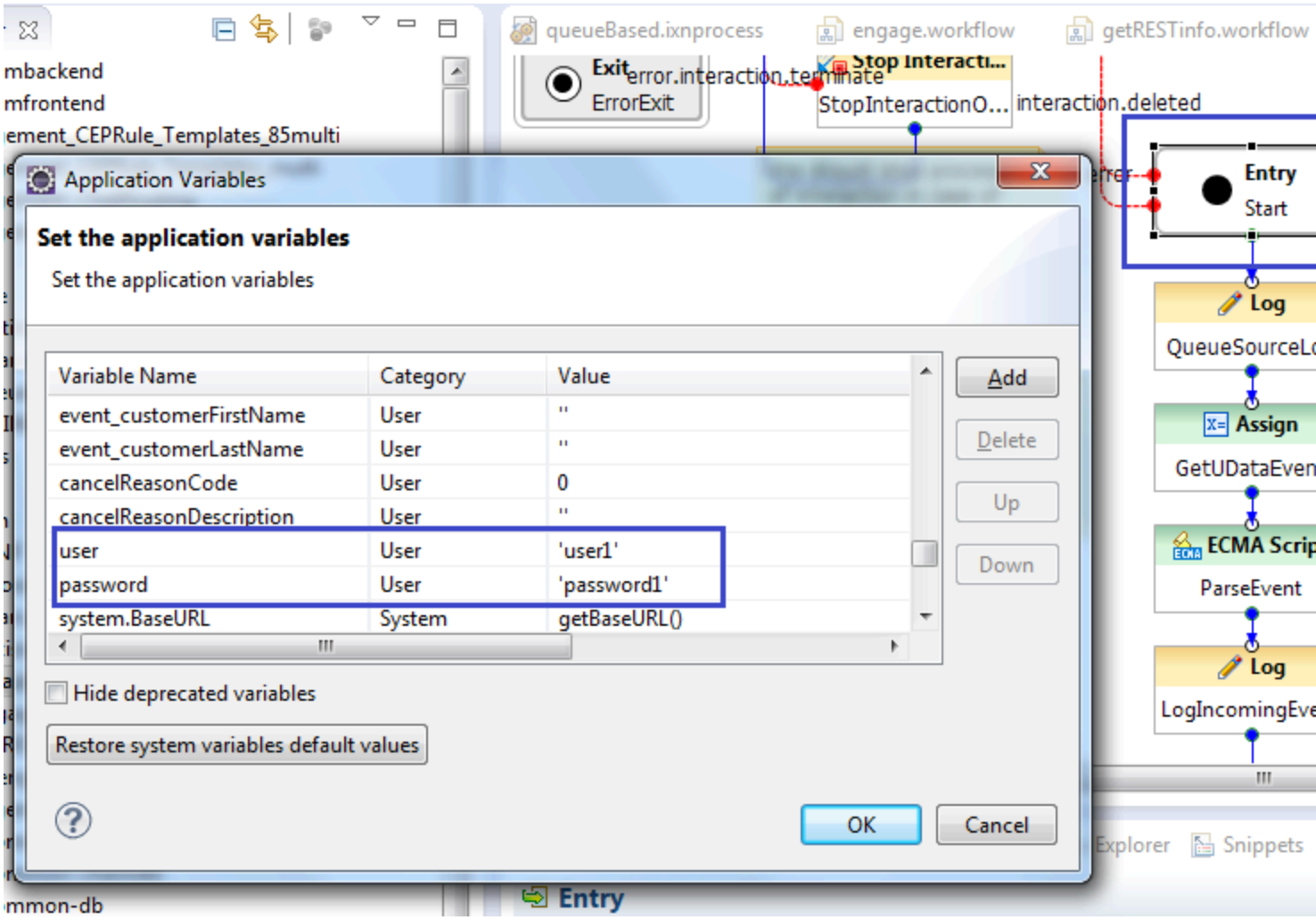
Genesys Web Engagement provides basic access authentication on the base of providing username/password pairs.

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



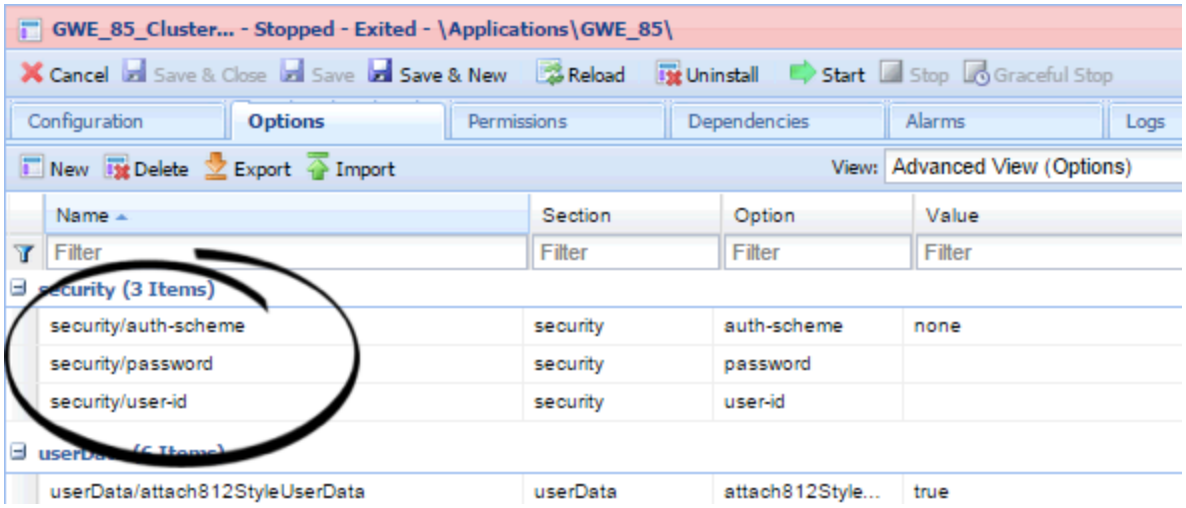
The username and password application variables in **getRESTInfo.workflow**.

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 the credentials specified in the security section of the Web Engagement Cluster or Web Engagement Server options:



The username and password are specified in the security section

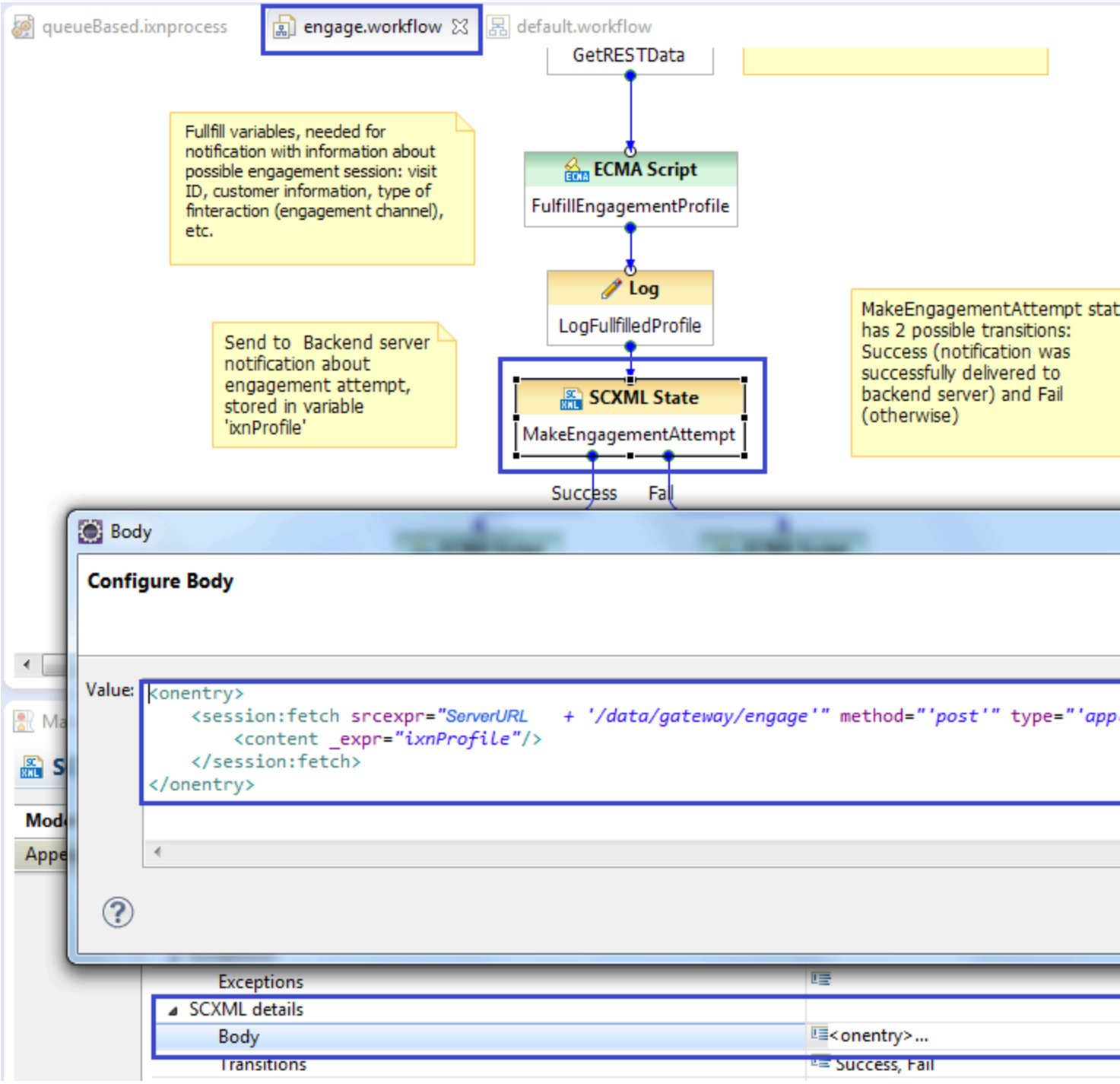
See Configuring Authentication for details.

Start Engagement as a Result of the Engagement Logic Strategy

Sending the "start engage" Request to the 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:



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

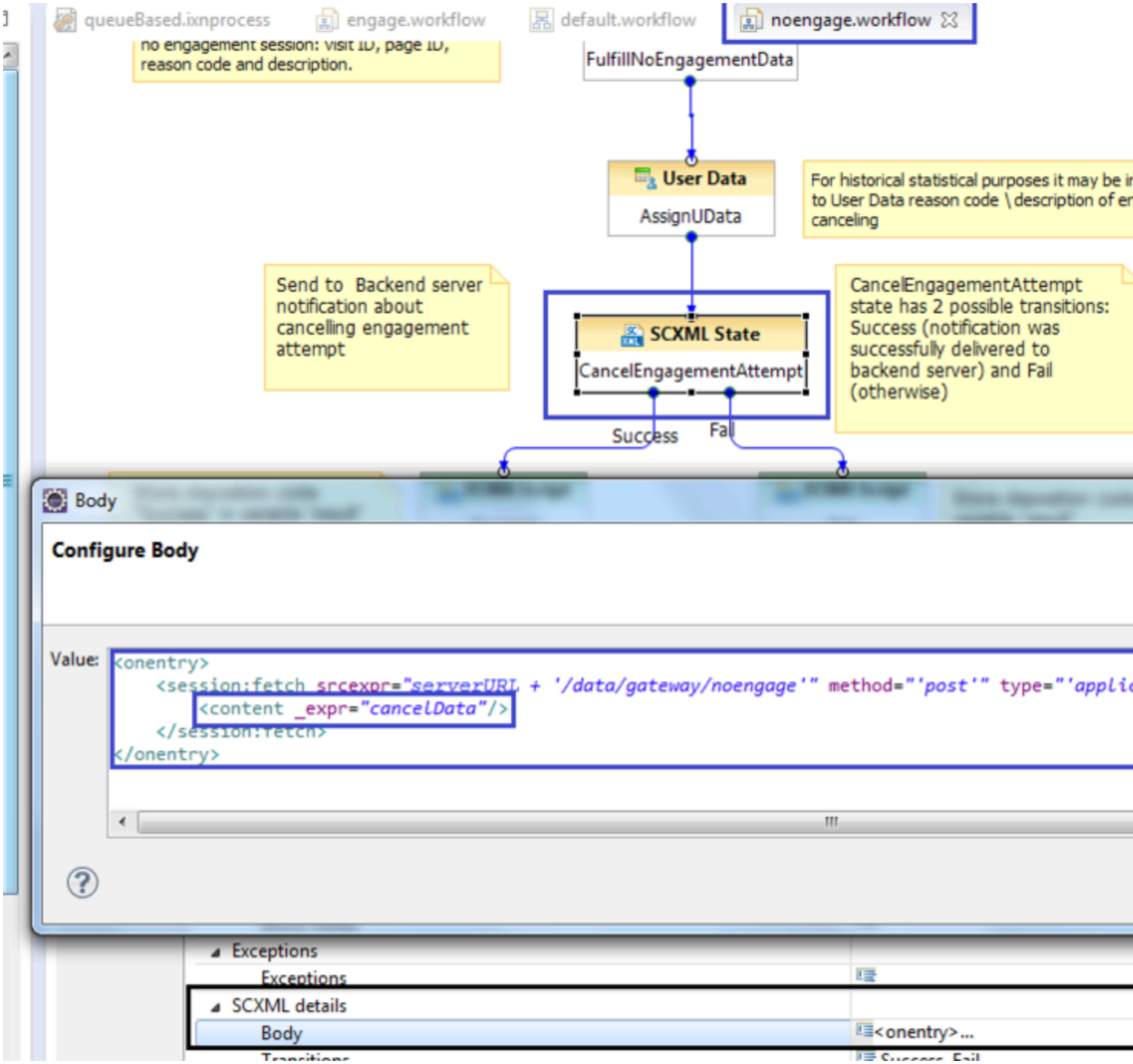
Field name	Field contents	Description
rule	String	Name of rule, which triggered this HotLeadActionableEvent
userID	String	String, which allows to identify authorized and recognized visitors For anonymous users it will be null
userState	String	State of current visit: Anonymous, Recognized or Authorized
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:

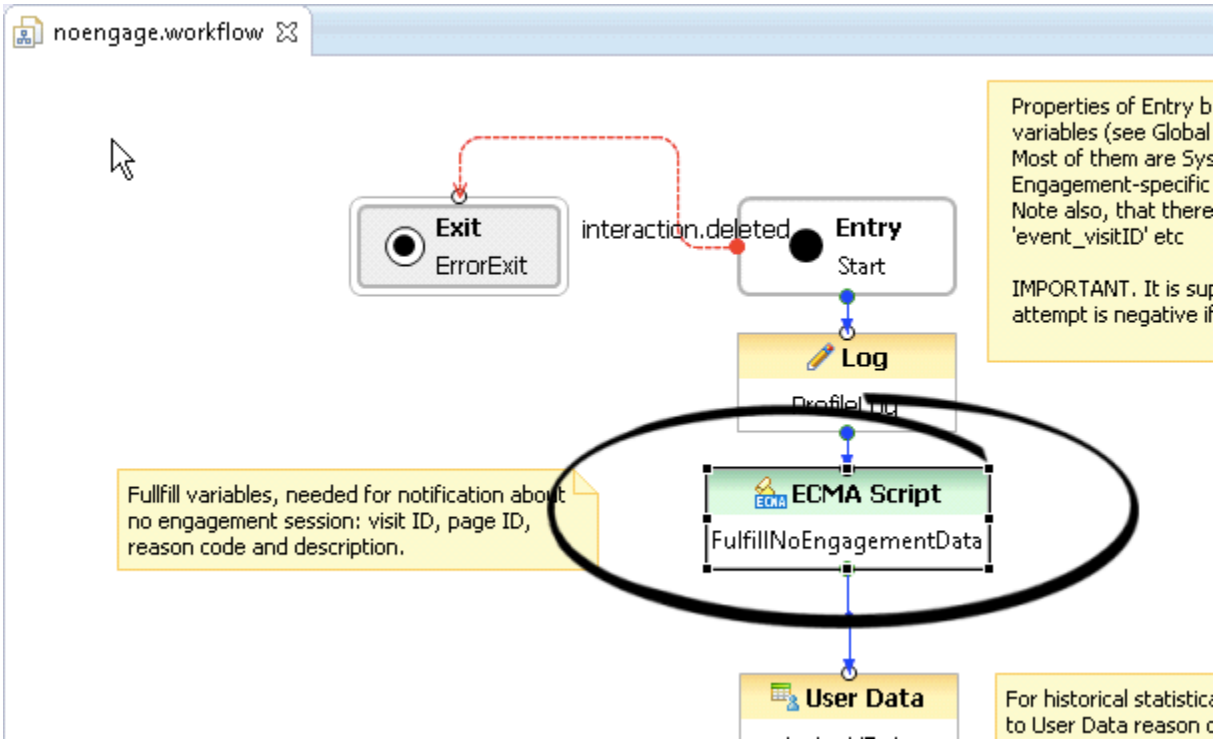


The REST request cancels the engagement

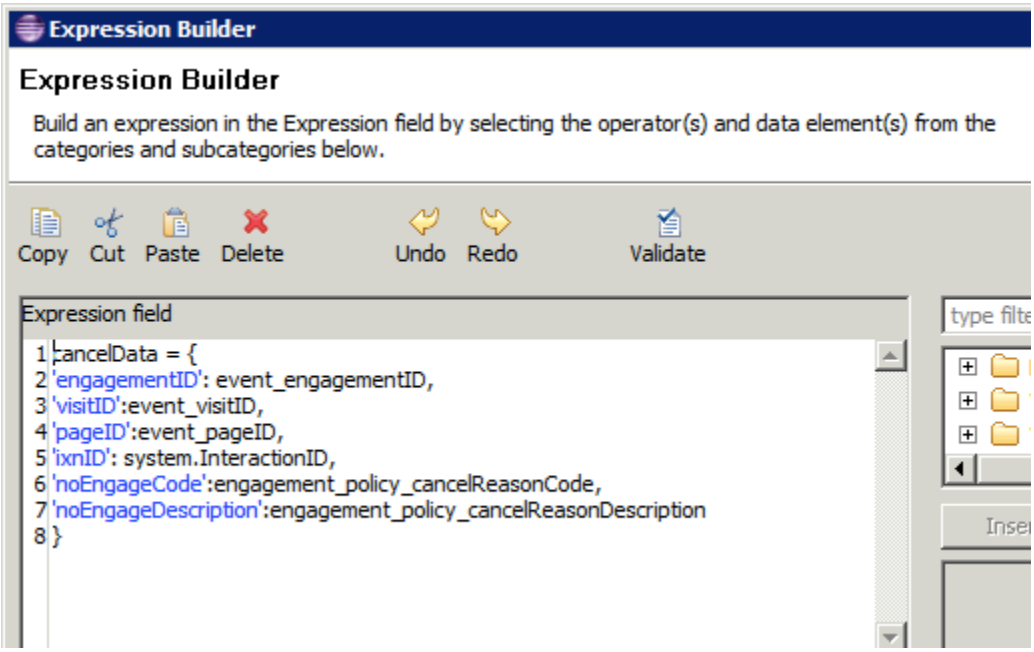
Security (authentication) aspects are the same as described in the **getRESTInfo.workflow**.

Fulfilling "no engage" Data

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



It contains six mandatory fields:



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.

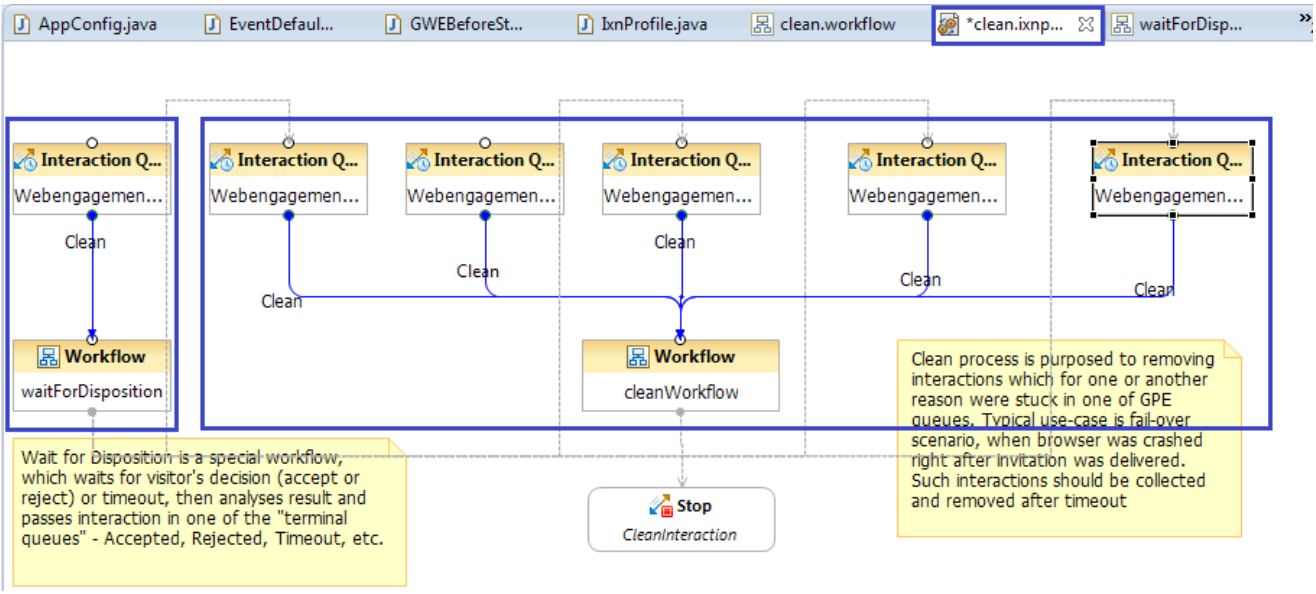
In this case, you need to define the cleaning process, which is also built on the top of ORS strategies.

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



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:

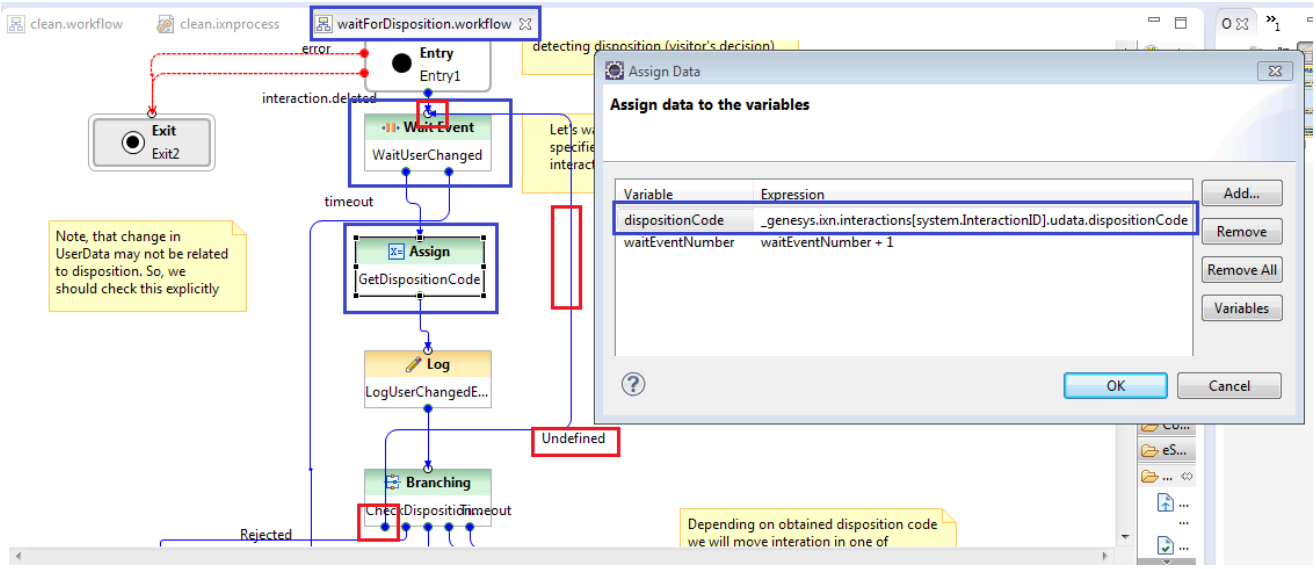
- **waitForDisposition.workflow**
- **clean.workflow**

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.

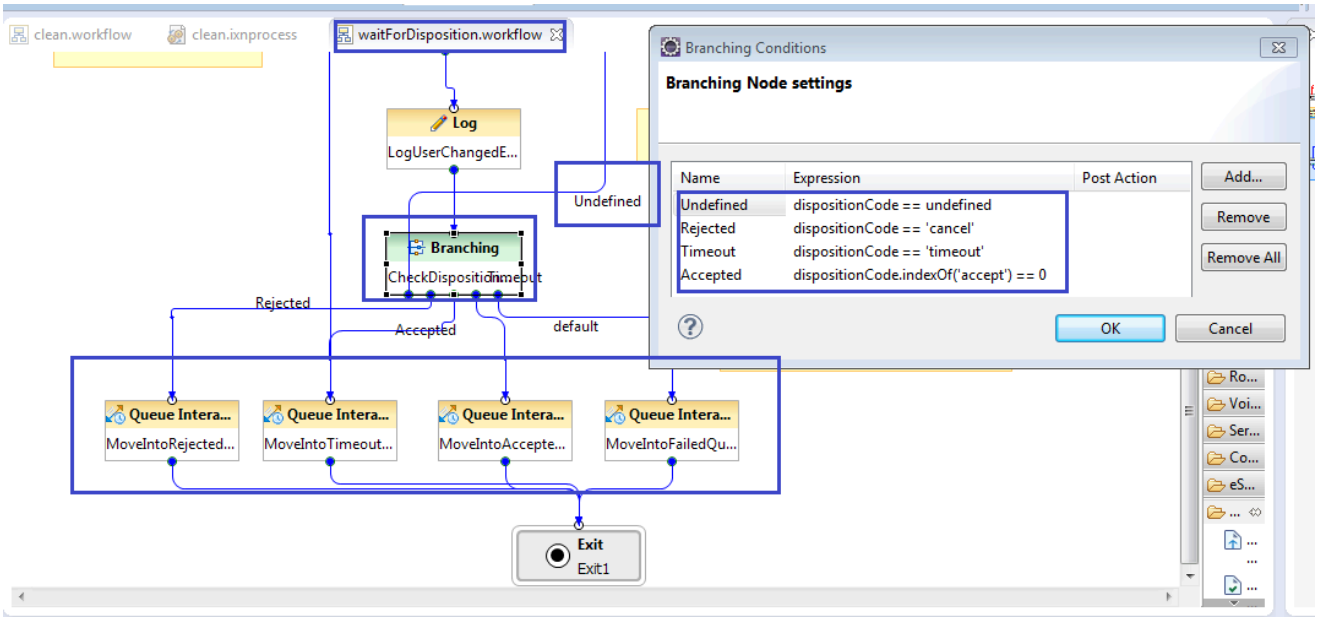


Here are the valid values for `dispositionCode` and the queues their interactions are placed in:

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.



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.

Customizing the Chat Routing Strategy

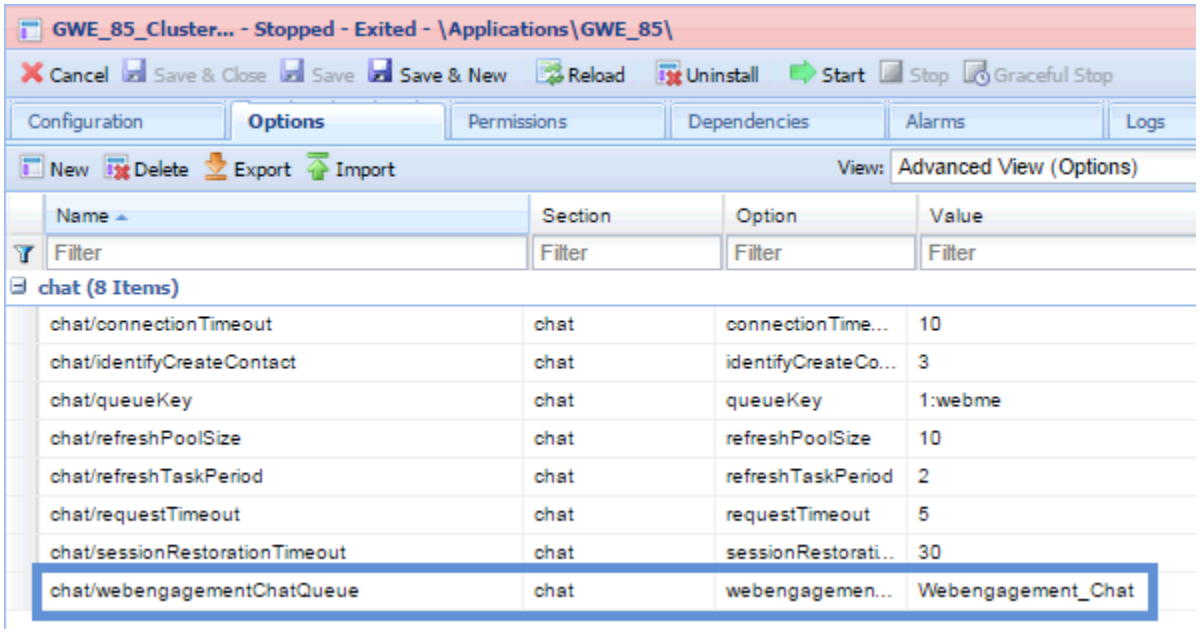
When you create your Web Engagement application, Genesys Web Engagement also creates default Engagement Logic and Chat Routing SCXML strategies in the **\apps*application_name*\resources_composer-projects** folder. Orchestration Server (ORS) uses these strategies to decide whether and when to make a proactive offer and how to route a chat interaction to a specific target.

The default Chat Routing strategy delivers chat interactions that are initiated in Genesys Web Engagement to a specific target. Although this strategy is included as part of the Web Engagement installation, it is possible to use your own existing strategy for routing. For example, a URS-based chat routing strategy; however, in this scenario you will need to adjust the Web Engagement solution to support the pacing algorithm functionality.

You can modify the Chat Routing SCXML by importing the Composer project into Composer. The project is located here: **\apps*application name*\resources_composer-projects\WebEngagement_ChatRouting**. Refer to the sections below for details about the Chat Routing strategy and how it can be modified.

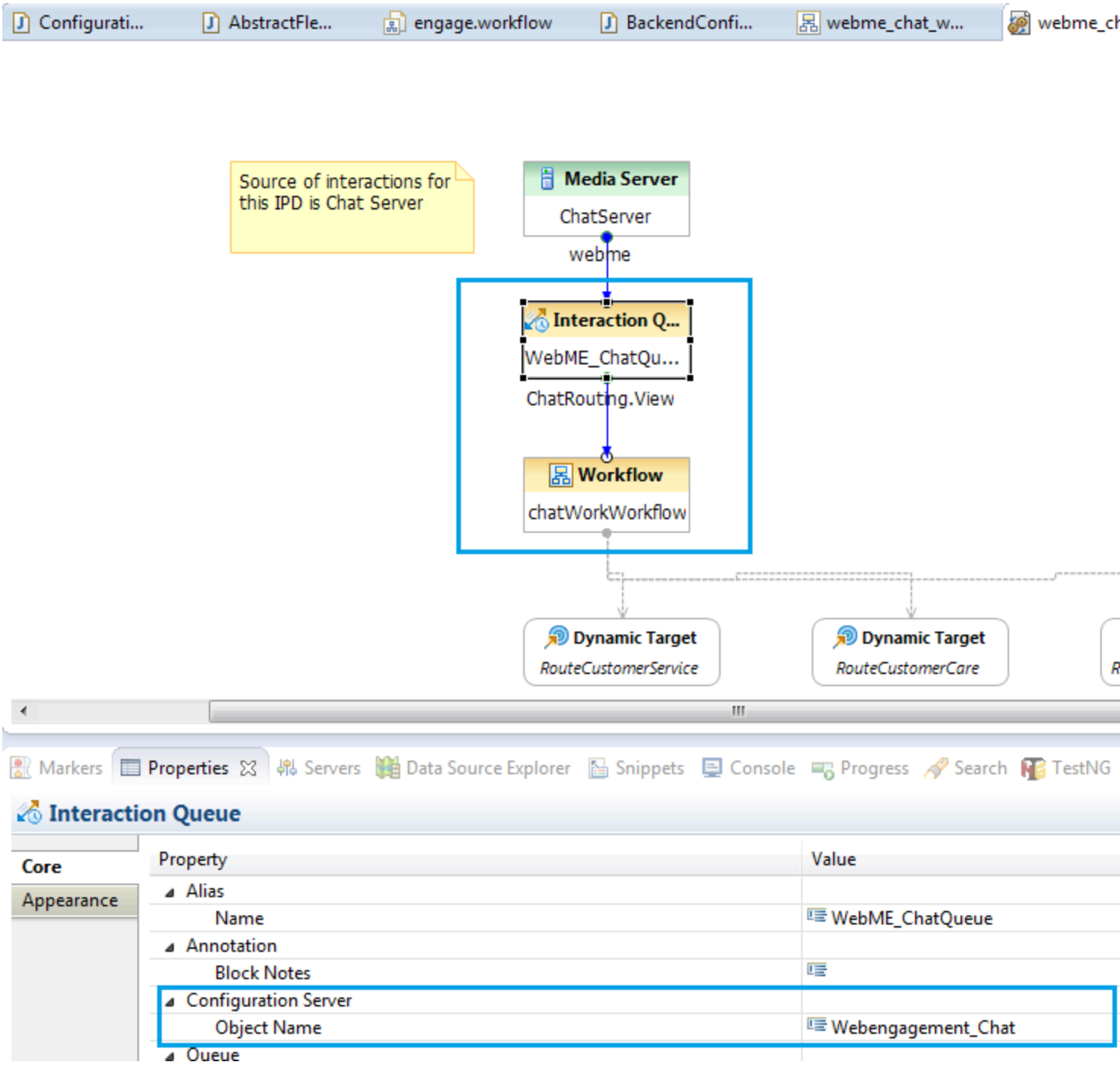
Main Interaction Workflow

The default entry point to the GWE Chat Routing strategy is the Interaction Queue specified in the webengagementChatQueue option on the Web Engagement Cluster application.



The Interaction Queue.

The interaction process pulls interactions from this queue and sends them through the chat workflow:



The chat workflow

Important

If you decide to change the value of **queueWebengagement**, make sure to also

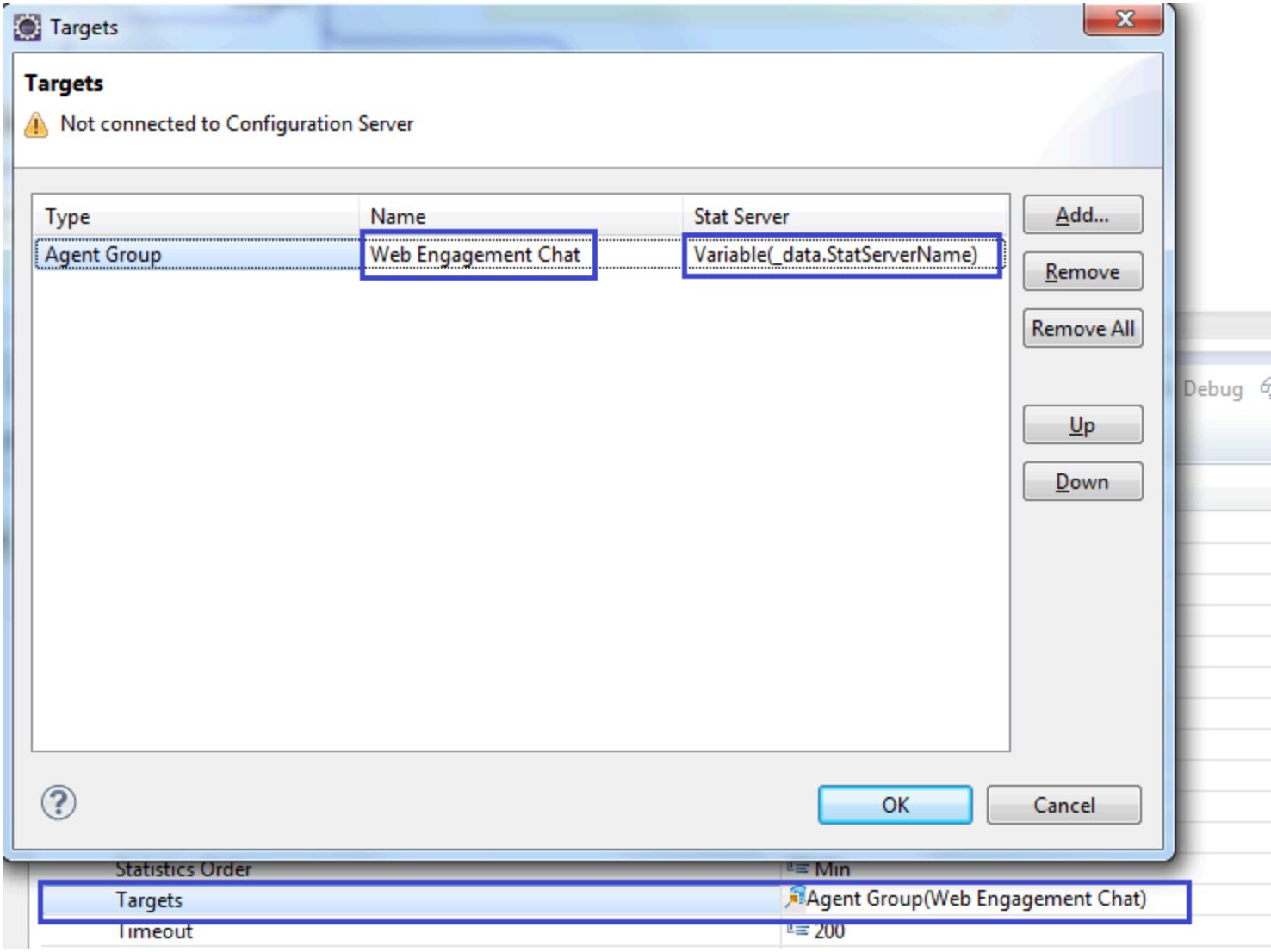
> adjust the name of the queue in the Chat Routing strategy.

The default Chat Routing strategy is straightforward and includes the following highlights in the workflow:

1. Obtain information from the User Data of the chat interaction that is being routed. See the **AssignCategory** block in the Chat Routing Strategy for details.
2. Send messages to the chat session from the routing strategy. See Sending Messages from the Chat Routing Strategy into the Chat Session for details.
3. Branch the workflow based on categories obtained from the chat interaction User Data. See the **BranchingByCategory** block for details.
4. Route to skill-based Virtual Groups. See the **RouteCustomerServer** and **RouteCustomerCare** blocks for details.
5. Route to a static Agent Group. See Routing to a Static Agent Group for details.

Routing to a Static Agent Group

When you plan to route an interaction to a static Agent Group, you should specify the name of this group and the name of the Stat Server in the Target property of the **RouteInteractionDefault** block.



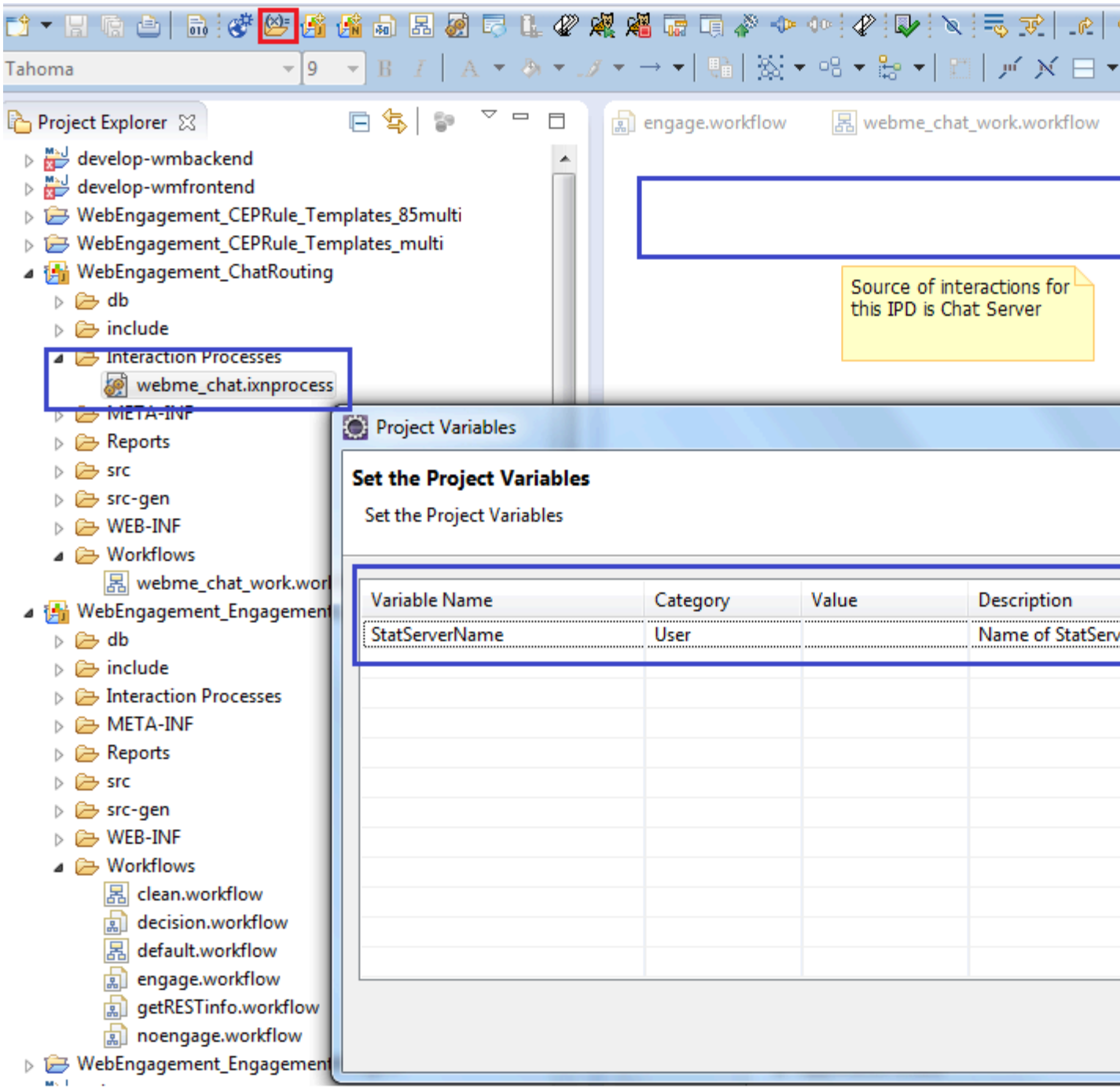
The Target property of the **RouteInteractionDefault** block.

In the image above, the Stat Server name is specified through the `Variable(_data.StatServerName)` variable. You can define this variable, or others like it, in Composer and Genesys Administrator.

Specifying Variables in Composer

Start

1. Double click the interaction process - in this case, `webme_chat.ixnprocess`.
2. Make sure that there are no elements selected in the opened interaction process.
3. Access the interaction process variables by clicking "Access Project Variables", marked with a red square in the image below:



Access the project variables

In the image above, the `StatServerName` variable is used in the default Chat Routing strategy.

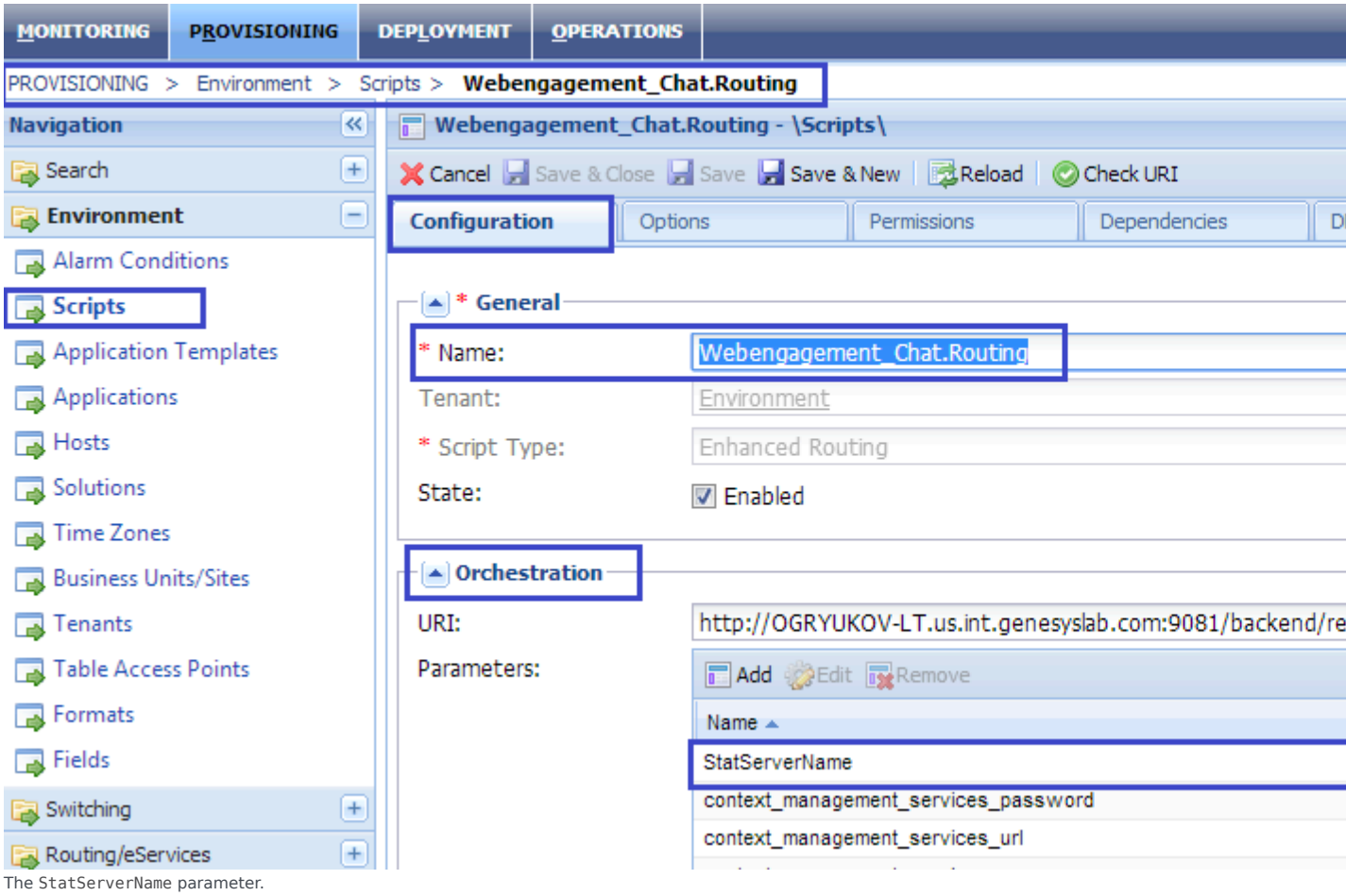
End

Specifying Variables in Genesys Administrator

The `StatServerName` parameter is set automatically by the Provisioning Tool when you install Genesys Web Engagement, but it can be changed manually.

Start

1. Navigate to `Provisioning > Environment > Scripts` and find the script with the entry-point Interaction Queue. In this case, the script is `Webengagement_Chat.Routing`.
2. In the Configuration tab, open the Orchestration section.
3. Now you can see a list of parameters that are passed into the Chat Routing strategy, including `StatServerName`.



The `StatServerName` parameter.

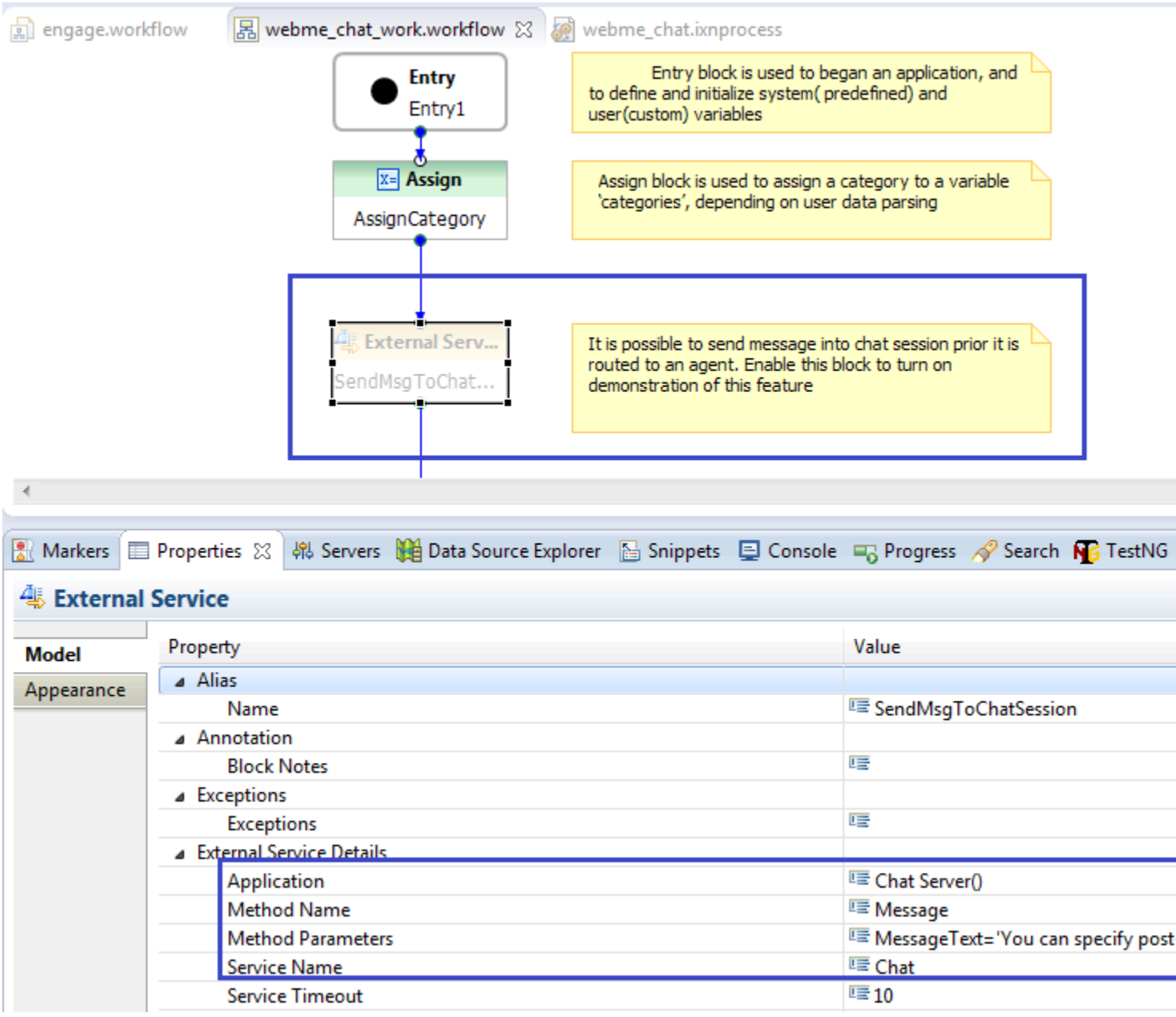
End

Sending Messages from the Chat Routing Strategy into the Chat

Session

There are times when you might need to send messages into the chat session directly from the routing strategy. For example, this could be additional information messages, advertising messages, and so on.

The default Chat Routing strategy contains an **External Service** block that provides this functionality:



The External Services block lets you send message from the routing strategy.

> **Important**
> The **External Service** block is disabled by default.

Customizing the Browser Tier Widgets

Deprecation notice

- **Starting with the 8.5.000.38 release of Genesys Web Engagement, Genesys is deprecating the Native Chat and Callback Widgets—and the associated APIs (the Common Component Library)—in preparation for discontinuing them.**

  This functionality is now available through a single set of consumer-facing digital channel APIs that are part of Genesys Mobile Services (GMS), and through Genesys Widgets, a set of productized widgets that are optimized for use with desktop and mobile web clients, and which are based on the GMS APIs.

  Genesys Widgets provide for an easy integration with Web Engagement, allowing you to proactively serve these widgets to your web-based customers.

  {{NoteFormat|Although the deprecated APIs and widgets will be supported for the life of the 8.5 release of Web Engagement, Genesys recommends that you move as soon as you can to the new APIs and to Genesys Widgets to ensure that your functionality is not affected when you migrate to the 9.0 release.

  - Note that this support for the Native Chat and Callback Widgets and the associated APIs will not include the addition of new features and that bug fixes will be limited to those that affect critical functionality.

Genesys Web Engagement includes pre-integrated Browser Tier widgets that are used for engagements. These widgets are based on HTML, CSS, and JavaScript, and can be customized to suit the look and feel of your website.

> ### Warning
> If you customize the widget HTML files, they will not be backward compatible with any new versions of Genesys Web Engagement.

Invitation Widget

Overview

The default invitation approach in Genesys Web Engagement is represented by **invite.html** (chat invitation). This HTML file, by default, has all the required dependencies embedded to avoid extra requests to the server.

The **invite.html** file has three code sections:

- Initial HTML Section
- JavaScript Third-party Libraries (dependencies) Section

- JavaScript Invitation Business Logic Section

Customization

There are four main ways you can customize the invitation widget:

- HTML/CSS — You can edit the HTML/CSS of your page or the **invite.html** file.
- Business Logic — You can modify the business logic included in the **invite.html** file to work with second- or third-party integration.
- Notification Service — You can change the JavaScript configuration through the Notification Service REST API.
- You can build your own version of the invitation. The default invitation widget is an example of a custom-built widget.

HTML/CSS

> **Important**
> In the paragraphs below, Genesys assumes that you have basic knowledge of CSS and HTML technologies.

If you need to change the basic style of the invitation (color, company logo, size, and so on) Genesys recommends that you use the HTML/CSS approach.

By default, the invite widget also contains all of the CSS needed for invite rendering, which is automatically added to the beginning of the <head> section of the web page when the invite is initialized.

If you need to modify these default styles, but you don't want to make it difficult to upgrade to newer versions of the widget, you can can create custom override styles. Make sure that your overrides are scoped to the components that need additional styling, and structure them so that they don't conflict with or overwrite any stand CSS files.

When overriding styles, consider the following points:

1. Review how the classes are assigned in the invite widget markup to better understand how they're applied (and how they can be overridden).
2. Create an override stylesheet. The best way to safely fine-tune a widget's appearance is to write new style rules that override the invite widget's styles and append these "override rules" in a separate stylesheet. Override rules are written against widget CSS class names and must appear in the source order after your theme stylesheet; since styles are read in order, the last style rule always takes precedence. By maintaining override styles in a separate file, you can customize the widget styles as much or as little as you'd like and still preserve the ability to easily upgrade the widget files as needed and simply overwrite your existing theme stylesheet, knowing that your override rules remain intact. Override rules can be listed in a dedicated stylesheet for overriding default website styles, or if you prefer to limit the number of files linked to your pages (and therefore limit the number of requests to the server), append override rules to the master stylesheet for your entire project.

To see exactly what you can override, you can use the developer tools that are commonly found in most modern web browsers. Currently, the Web Engagement CSS selector is not documented and there is no guarantee for backward compatibility for future versions of the invite widget.

Customization Examples

Change subject, message, buttons caption

```
<div title="New Subject" class="gpe-helper-hidden gpe-dialog">
    <div class="gpe-branding-logo"></div>
    <div class="my-message-content">
        <span>New Message</span>
    </div>
</div>
```

Change colors (message, subject, background, and so on)
For example, if you want to change the dialog style to red colors, you can add these styles to your page:

```
<style>
    .gpe-dialog .gpe-dialog-titlebar {
        background-color: red;
    }
    .gpe-dialog .gpe-button-text {
        color: red;
    }
</style>
```

Or message color:

```
<style>
    .gpe-dialog .message-content {
        color: blue;
        background-color: red;
    }
</style>
```

Or inline customization:

```
<div title="Chat" class="gpe-helper-hidden gpe-dialog">
    <div class="gpe-branding-logo"></div>
    <div class="message-content" style="color: #ffcc00; background-color: #0066ff
"></div>
</div>
```

Size of Invitation Widget
To change the size (width and height) of the invite widget, you can use following snippet:

```
<style>
    .gpe-dialog {
        width: 300px !important;
        height: 200px !important;
    }
</style>
```

Branding Logo
To customize the branding logo, you can use the CSS class "gpe-branding-logo". By default, the **invite.html** file uses an embedded image resource with a Data URI Scheme (http://en.wikipedia.org/wiki/Data_URI_scheme) in base64 format:

```
<div class="gpe-branding-logo" style="
    background-image: url(  ...  AAASUVORK5CYII=);
">
```

To customize the logo, you can generate the same base64 data code for your own image with the generator (http://base64converter.com/).

Alternatively, you can just use CSS:

```
<div class="branding-content" style=" background-image:url('myLogo.png'); "></div>
```

Business Logic

The **invite.html** file includes functions that you can change or replace for second- or third-party media integration:

- `init()`
- `startChat()`
- `sendInviteResult()`
- `onAccept()`

Generally, you will need to work inside the `startChat()` function, but you can also make additional changes in other functions. For example, if you need to integrate another type of media besides chat, you can use `onAccept()` to extend the number of medias the invite supports. You must also be sure to make any necessary changes in the Engagement Logic Strategy.

Customization Examples

- Integration with Second- and Third-Party Media - Examples

Notification Service

The Notification Service is used to pass data to the invitation from the server. By default, data is composed in the **engage.workflow** of the Engagement Logic Strategy (**apps/*application name*/resources/_composer-projects/WebEngagement_EngagementLogic/Workflows/engage.workflow**).

You can use predefined commands in the Notification Service REST API to show your own invitation — particularly, gpe.callFunction and gpe.appendContent.

Customization Examples

- Notification Service REST API - Using the API to Customize Widgets

Localization

You can localize the invite by using the Chat Invitation Message. Use the **subject**, **message**, **acceptBtnCaption** and **cancelBtnCaption** options to set specific text for the invite widget.

Chat Widget

Overview

The chat widget provides the main chat functionality for Genesys Web Engagement. It's a versatile widget that can be customized through the Chat Service JS API and the Chat Widget JS API.

Customization

There are three different customization types available for modifying the chat widget UI: Template-based, CSS-based, and JavaScript-based. Using these customization types, you can do any of the following:

- modify the structure of the widget
- add content
- add css classes
- modify the style (including the logo and buttons)
- use JavaScript UI hooks to modify the widget

For details about the customization types and how you can use them, see Customizing the User Interface, part of the Chat Widget JS API.

You can also use the Chat Service JS API to build your own chat widget and control chat sessions. Before creating your own chat widget, be sure to review the default chat widget — it's highly customizable through the Chat Widget JS API, and it also provides access to the same Chat Service JS API.

Localization

You can use the `startChat` and `restoreChat` methods of the Chat Widget JS API to enable localization for the chat widget. For details and step-by-step instructions, see Localization.

Deploying an Application

Complete the procedures on this page after you have created your Genesys Web Engagement application.

Deploying your Application

> ### Warning
> You must only deploy an application when the GWE servers are running.

Prerequisites

- You have created your app.

Start

1. Navigate to the installation directory for Genesys Web Engagement and open a new console window.
2. Use the `deploy` script (`deploy.bat` on Windows and `deploy.sh` on Linux) to deploy your application:

   ```
   deploy <your_application_name>
   ```

   Note: To request debug-level logs while this command is executed, use the **-v** parameter. For example:

   ```
   deploy myApp -v
   ```

End

Note that you will be prompted to enter your username and password to receive write-access to the Config Server if your application contains category definitions. This doesn't apply for new applications you create, but will for the *playground* application as it contains pre-defined categories.

The deploy script copies files to the appropriate locations. If the deploy is successful, the script output displays a `DEPLOY SUCCESS` messages at run-time.

Next Steps

- Deploying a Rules Package

Deploying a Rules Package

Creating a rules package is the final step before you are ready to test your new application. Refer to the Application Development Tasks for details about the previous steps.

Rules are mandatory for managing actionable events generated from the System and Business event flows submitted by the Browser Tier. To add rules, you must create a package and then a set of rules. For details about rules, refer to the Genesys Rules System documentation.

After completing the steps on this page, the rules are deployed to the Web Engagement Servers.

Complete the following steps to create and deploy a rules package:

1. If you need to map your rules to a particular domain, review Multi-Package Domain Oriented Rules.
2. Creating a Rules Package
3. Creating Rules in the Rules Package
4. Deploying the Rules Package

Multi-Package Domain Oriented Rules

Genesys Web Engagement supports multi-package domain oriented rules. You can map your rules package to a particular domain by reversing the domain zone in the name of the rules package. For example, the `blog.genesys.com` domain would have a rules package called `com.genesys.blog`.

You can have multiple rules packages on the same server at the same time. New rules packages (with a different package name) that are deployed do not rewrite the current rules, but are instead added to the current rules set. When the existing rules package is deployed, it rewrites selected package rules in the current rules set.

This domain mapping is applied hierarchically - the "root" domain is processed by the "root" package and the sub-domain is process by the sub-package and all parent packaged (including "root").

For example, your website contains the following sub-domains:

- genesys.com
- blog.genesys.com
- communication.genesys.com
- personal.communication.genesys.com

And you have the following rules packages:

- com.genesys

- com.genesys.blog
- com.genesys.communication
- com.genesys.communication.personal

The rules packages are processed as follows:

Domain	com.genesys	com.genesys. blog	com.genesys. communication	com.genesys. communication. personal
genesys.com	+	-	-	-
blog.genesys.com	+	+	-	-
communication.genesys.com	+	-	+	-
personal.communication.genesys.com	+	-	+	+

> **Important**
>
> This feature is turned off by default. You can turn on domain separation rule execution on the specified Web Engagement server by setting the domainSeparation option to `true`.

Creating a Rules Package

Complete the steps below to create the rules package associated with your Web Engagement application. This procedure is an example of how to create a rules package. For further information about creating rules, refer to the Genesys Rules System Deployment Guide.

Prerequisites

- Your environment includes Genesys Rules Authoring Tool. See Genesys environment prerequisites for compliant versions.
- Roles are configured to enable your user to create rules.
- Your CEP Rule templates were published.

Start

1. Open the Genesys Rules Authoring Tool and navigate to `Environment > Solution > New Rule Package`.
2. In the `General` tab:
   - Enter a `Package Name`. For example, `myproject.rules.products`.
   - Enter a `Business Name`. For example, `Products`.
   - Select `web_engagement` for `Package Type`. `WebEngagement_CEPRule_Templates` appears in the `Template` table.
   - Optionally, you can enter a `Description`.
3. Select `WebEngagement_CEPRule_Templates` in the `Template` table.

Create a new rules package

4. Click Save.

End

Creating Rules in the Rules Package

Prerequisites

- Creating a Rules Package

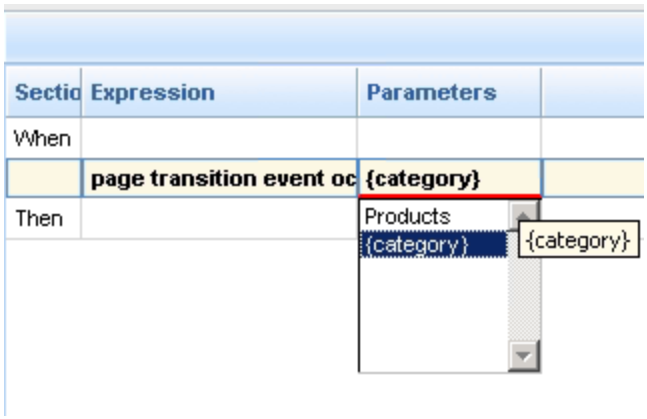
Start

1. In Genesys Rules Authoring Tool, select the rules package you created in the previous procedure.
2. Select the `Rules` tab.
3. Click `New Linear Rule`. This creates a new rule in the `Rules` table.
   Note: Web Engagement does not support GRAT Decision Tables. You must only use Linear Rules.

4. Select the created rule:
   - Enter a `Name`. For example, `Products`.
   - Enter a `Phase`. The list of rule phases can be modified by changing the values of the Phases enumeration in the CEP Rules Template. The default value is *.
5. Click `Add Condition`:
   - Scroll down to select a condition. For example, a `page transition event occurs and belongs to category`, which launches the actionable event any time that a user enters or leaves a page on your website.

Select your rule's condition

   - Select a category in `Parameters`. For example, `Products`. The `Parameters` list displays the categories that you previously created.



Set the condition's parameters

6. Click `Add Action` and select an action in the list. For example, `generate actionable event`.
7. Click `Save...`.
   You can create as many rules as you need in your rules package.

End

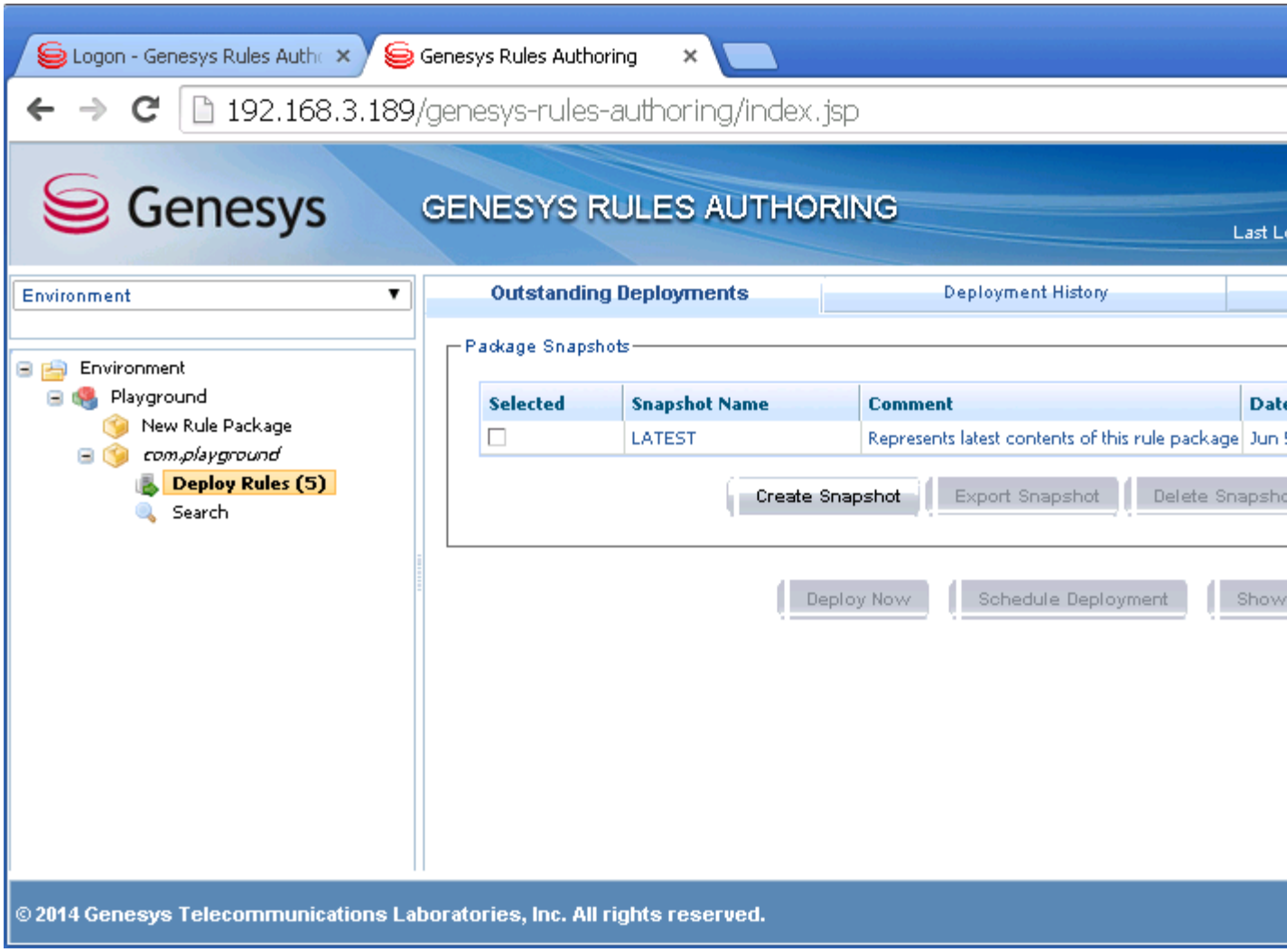
Deploying the Rules Package

Prerequisites

- Your GRAT application has a connection to the GWE Cluster application.
- You started the Web Engagement servers.

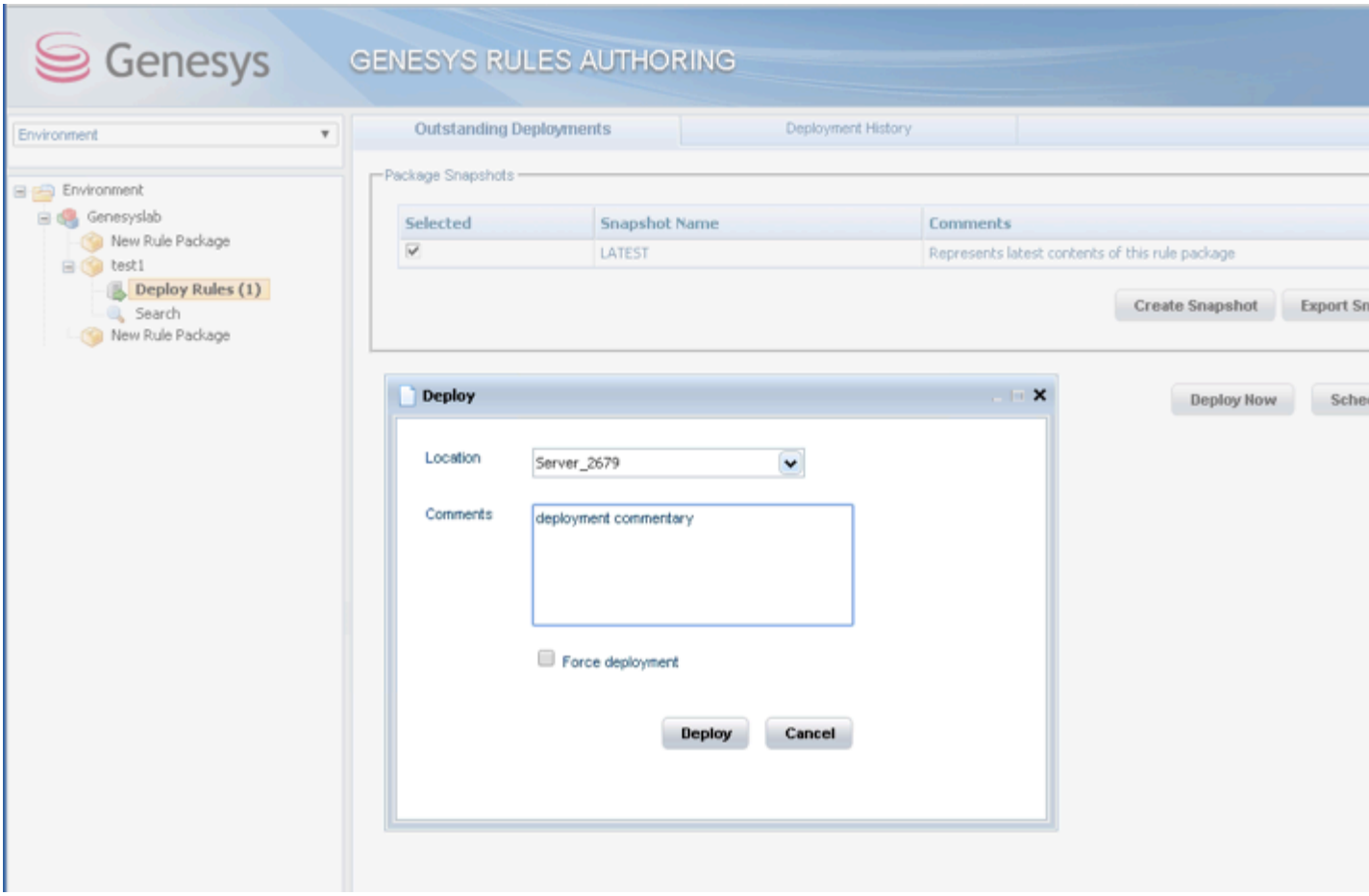
Start

1. In Genesys Rules Authoring Tool, navigate to **Solution** > ***your rules package*** > **Deploy Rules**.



Deploy the rules package

2. Select the checkbox next to your rules package in the Package Snapshots section.
3. Click **Deploy Now**. The **Deploy** window opens.
4. Select your Genesys Web Engagement Server for the **Location**.



Deploy the rules package

5. Click **Deploy**. The rules package is deployed to the Web Engagement system.

End

Next Steps

- If you are following the Lab deployment scenario, you can test your application with the ZAP Proxy.

Testing with ZAP Proxy

The ZAP Proxy is a development tool that allows you to test your application without adding the JavaScript tracking code to your website. Once you have configured this proxy, you can launch it and start the Genesys Web Engagement servers to start testing your application by emulating a visit on your website. In a few clicks, without modifying your website, Genesys Web Engagement features will show up on a set of web pages, according to the rules and categories that you created.

There are two proxy tools available in the Genesys Web Engagement installation: Simple and Advanced. See the appropriate tabs below for details and configuration information.

Simple ZAP Proxy

To use the Simple ZAP Proxy, you must first complete a few procedures to configure the tool and your web browser.

Getting the ZAP Proxy Port

Complete this procedure to retrieve the ZAP Proxy port, which you will need later when you configure your web browser.

Start

1. Navigate to **C:\Users*current user*\ZAPProxy**.
   If this folder does not exist, navigate to your Web Engagement installation directory and launch **proxy.bat** (on Windows) or **proxy.sh** (on Linux). The **ZAPProxy** folder appears automatically.
2. Edit **config.xml** and find the **<proxy>** tag.
3. Check that the value of the **<ip>** tag is set to your host IP address.
   Note: You cannot use 127.0.0.1 or localhost for this value.
4. Note the value of the **<port>** tag (usually 15001).
5. Save your changes.

End

Configuring the Proxy

> **Important**
>
> The proxy configuration file will appear after you deploy your Web Engagement application. Also, note that the **playground** application does not include a proxy configuration file (instead it contains the entire website).

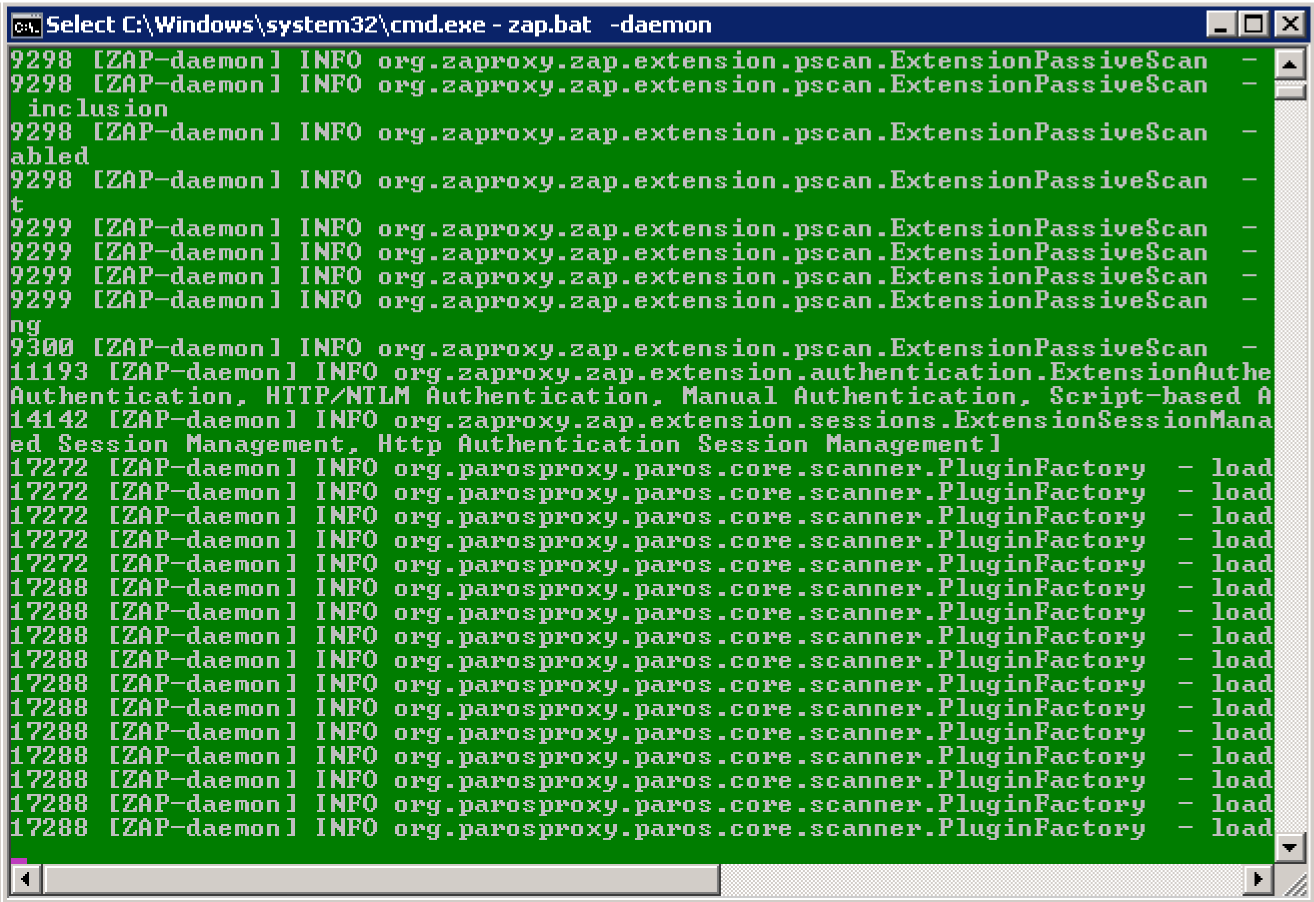
Start

1. Navigate to the **\tools\proxy\plugin** folder inside your your Web Engagement installation directory.
2. Open the configuration file, which is called **FilterMultiReplaceResponseBody.xml**.
3. Change **<enable>false</enable>** to **<enable>true</enable>**.

End

Starting the Proxy

Navigate to your Web Engagement installation directory and launch **proxy.bat** (on Windows) or **proxy.sh** (on Linux). The Simple ZAP Proxy starts.



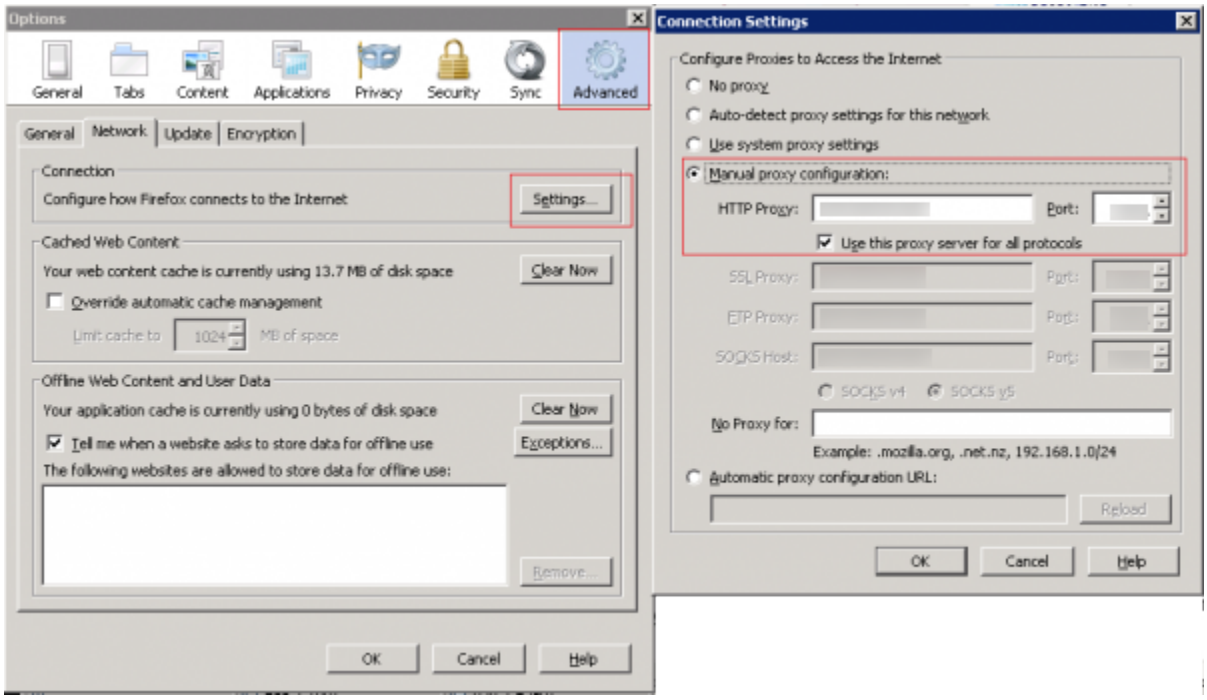
The Simple ZAP Proxy

Setting Up your Web Browser

Configure your web browser to use the Simple ZAP Proxy.

Start

1. Start your web browser.
2. Open your Internet settings. For instance, in Mozilla Firefox, select **Tools > Options**. The **Options** dialog window appears.
3. Select **Advanced**, and in the **Network** tab click **Settings...** The **Connection Settings** dialog windows appears.
4. Select the **Manual proxy configuration** option:
   - Enter your host IP address in the **HTTP** proxy text box.
   - Enter the port used by the ZAPProxy in the **Port** text box. This is the value you retrieved in Getting the ZAPProxy Port.
   - Select the option **Use this proxy server for all protocols**.



ZAPProxy used in Firefox

5. Click **OK**. Now your browser is set up for the ZAP Simple Proxy. To use the proxy, all you need to do is navigate to the site where you want the proxy to inject the Web Engagement instrumentation script and browse through the web pages.

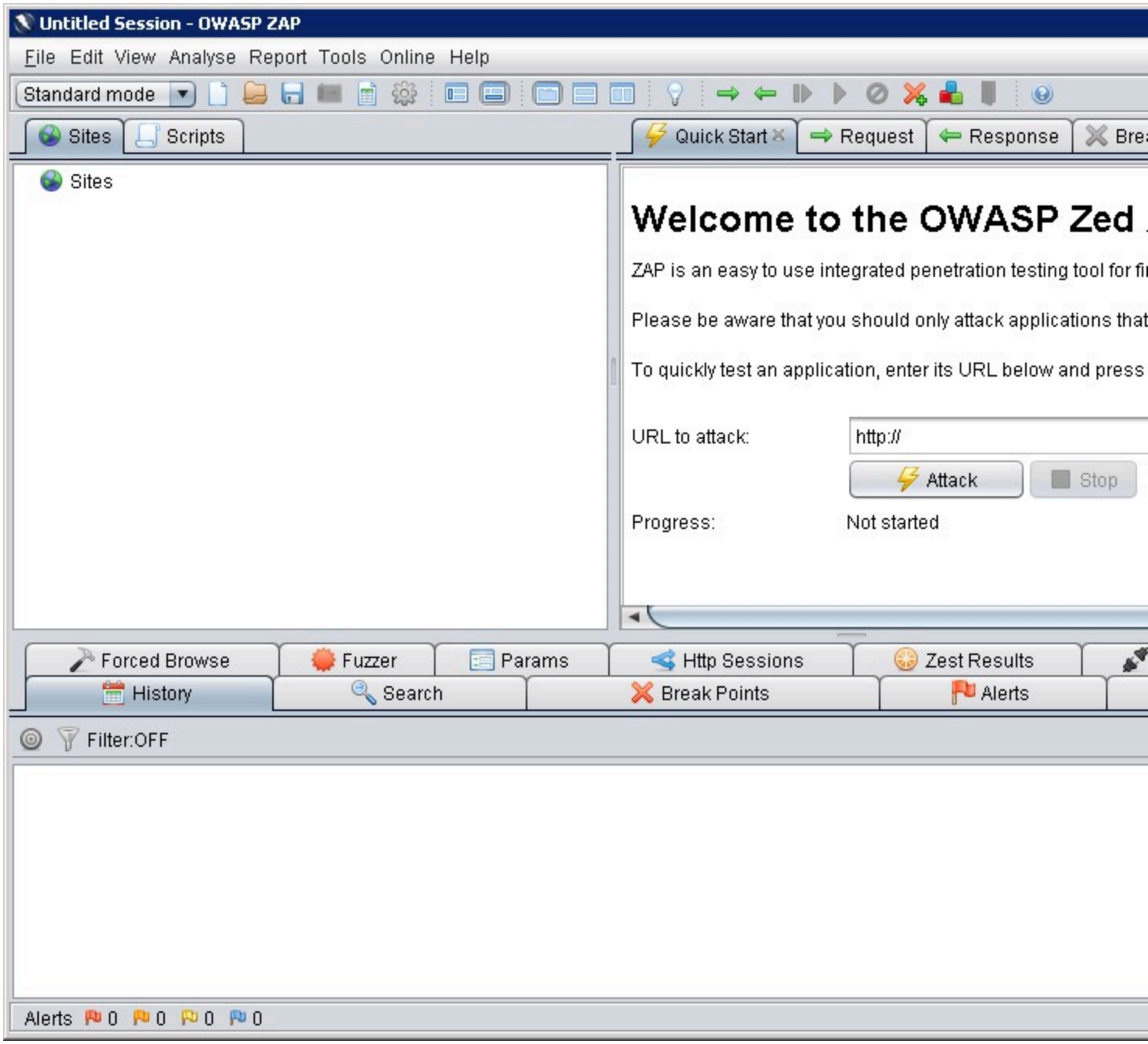
End

Advanced ZAP Proxy

The Advanced ZAP Proxy is based on the OWASP Zed Attack Proxy Project (ZAProxy). In addition to acting as a proxy, the Advanced ZAP Proxy also provides a UI and validates vulnerabilities in your website at the same time. To use the Advanced ZAP Proxy, you must first complete a few procedures to configure the tool.

Starting the Proxy

Navigate to your Web Engagement installation directory and launch **tools\proxy\zap.bat** (on Windows) or **tools\proxy\zap.sh** (on Linux). The proxy starts.



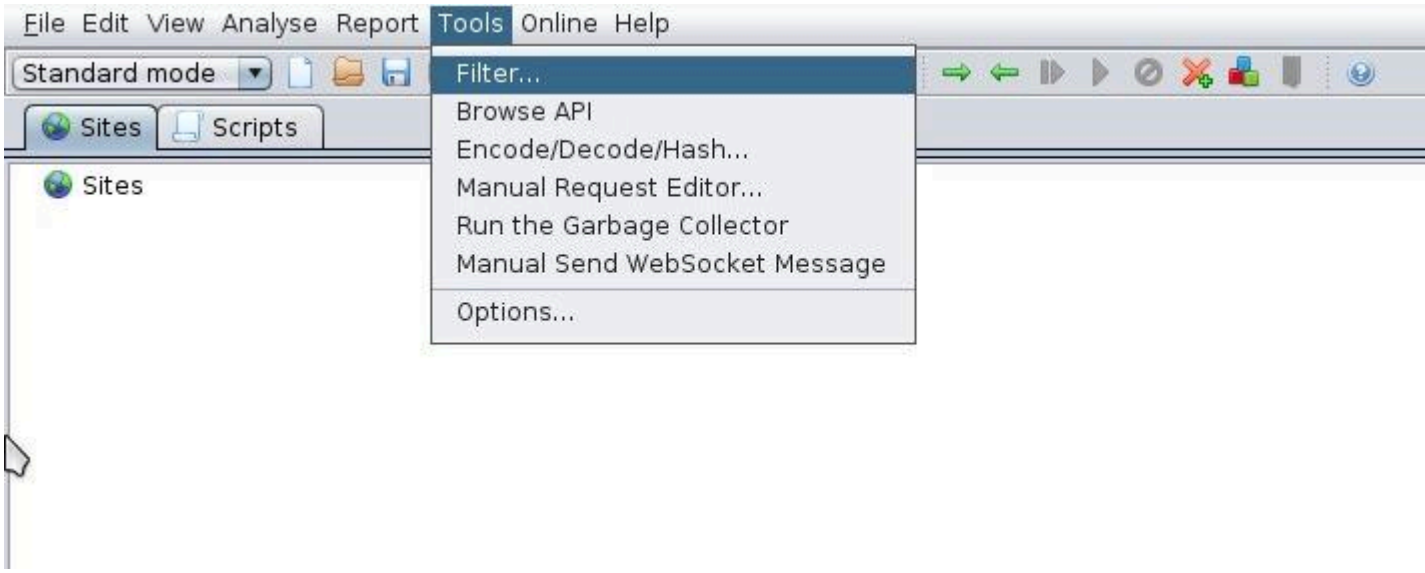
The Advanced ZAP Proxy

Configuring the Proxy

Once the proxy is running, you can configure it using the GUI.

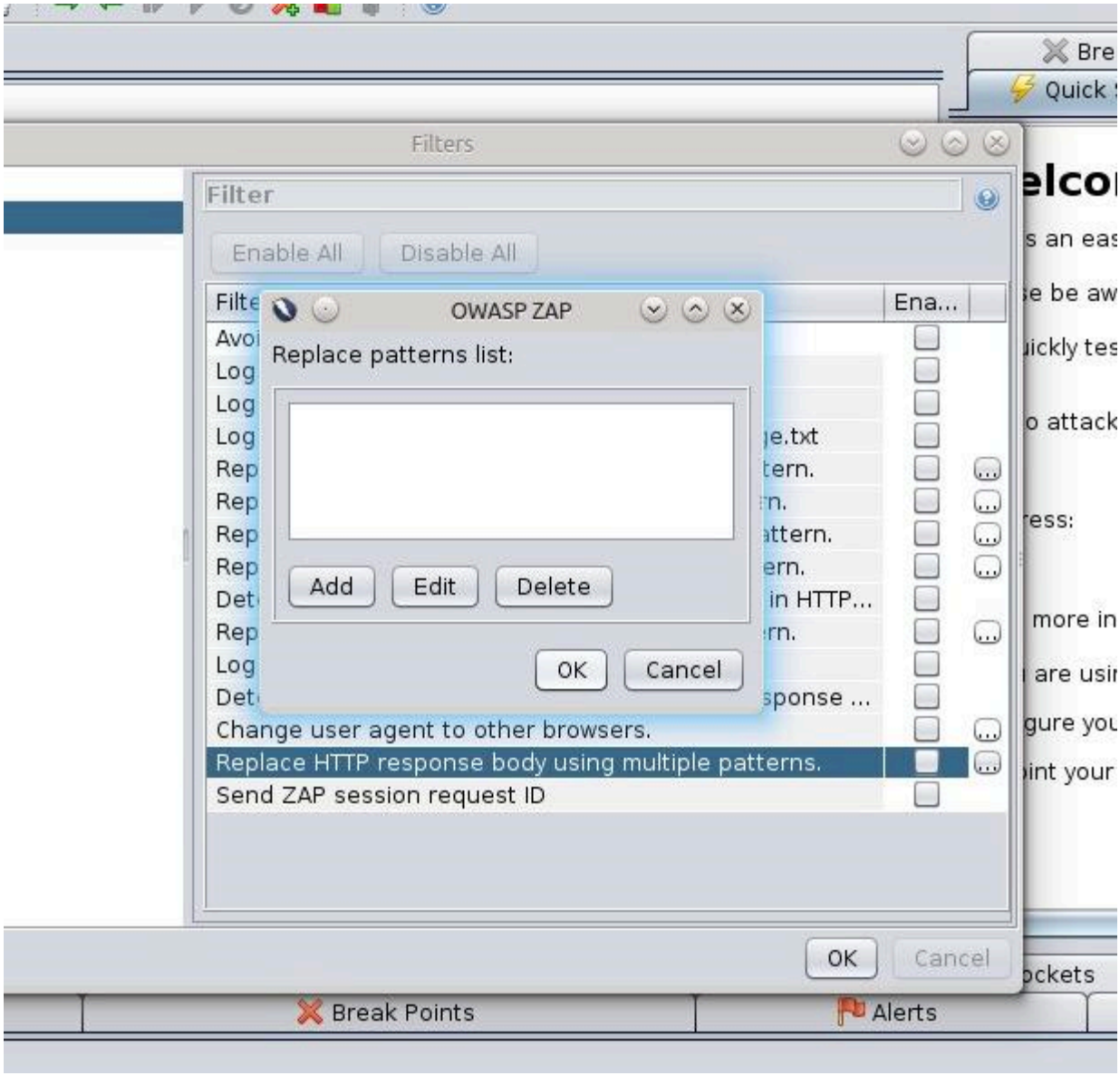
Start

1. Open **Tools > Filter...**.



Select the Filter menu item.

2. In the list of filters, select **Replace HTTP response body using multiple patterns** and click **...** to edit the filter.



Select the filter.

3. Click **Add** and enter the following information:
   - Pattern - **</head>**
   - Replace with -

```
<script>
 var _gt = _gt || [];
 _gt.push(['config', {
 dslResource: ( 'https:' == document.location.protocol ? 'https://<Web Engagement Server
 host>:<Web Engagement Server port>' :
 '<Web Engagement Server host>:<Web Engagement Server port>') + '/server/api/resources/v1/
 dsl/domain-model.xml',
 httpEndpoint: '<Web Engagement Server host>:<Web Engagement Server port>',
 httpsEndpoint: '<Web Engagement Server host>:<Web Engagement Server secure port>'
 }]);
```

```
var _gwc = {
widgetUrl: ( 'https:' == document.location.protocol ? 'https://<Web Engagement Server
host>:<Web Engagement Server port> :
'<Web Engagement Server host>:<Web Engagement Server port>') + '/server/api/resources/v1/
chatWidget.html'
};
(function(gpe, gwc) {
if (document.getElementById(gpe)) return;
var s = document.createElement('script'); s.id = gpe;
s.src = ( 'https:' == document.location.protocol ? 'https://<Web Engagement Server
host>:<Web Engagement Server port>' :
'<Web Engagement Server host>:<Web Engagement Server port>') + '/server/resources/js/
build/GPE.min.js';
s.setAttribute('data-gpe-var', gpe);
s.setAttribute('data-gwc-var', gwc);
(document.getElementsByTagName('head')[0] || document.body).appendChild(s);
})('_gt', '_gwc');
</script>
</head>
```

4. Click **OK** to save the pattern.
5. If you need to check or update the ZAP port address, open **Tools > Options...** and review the **Local proxy** section.

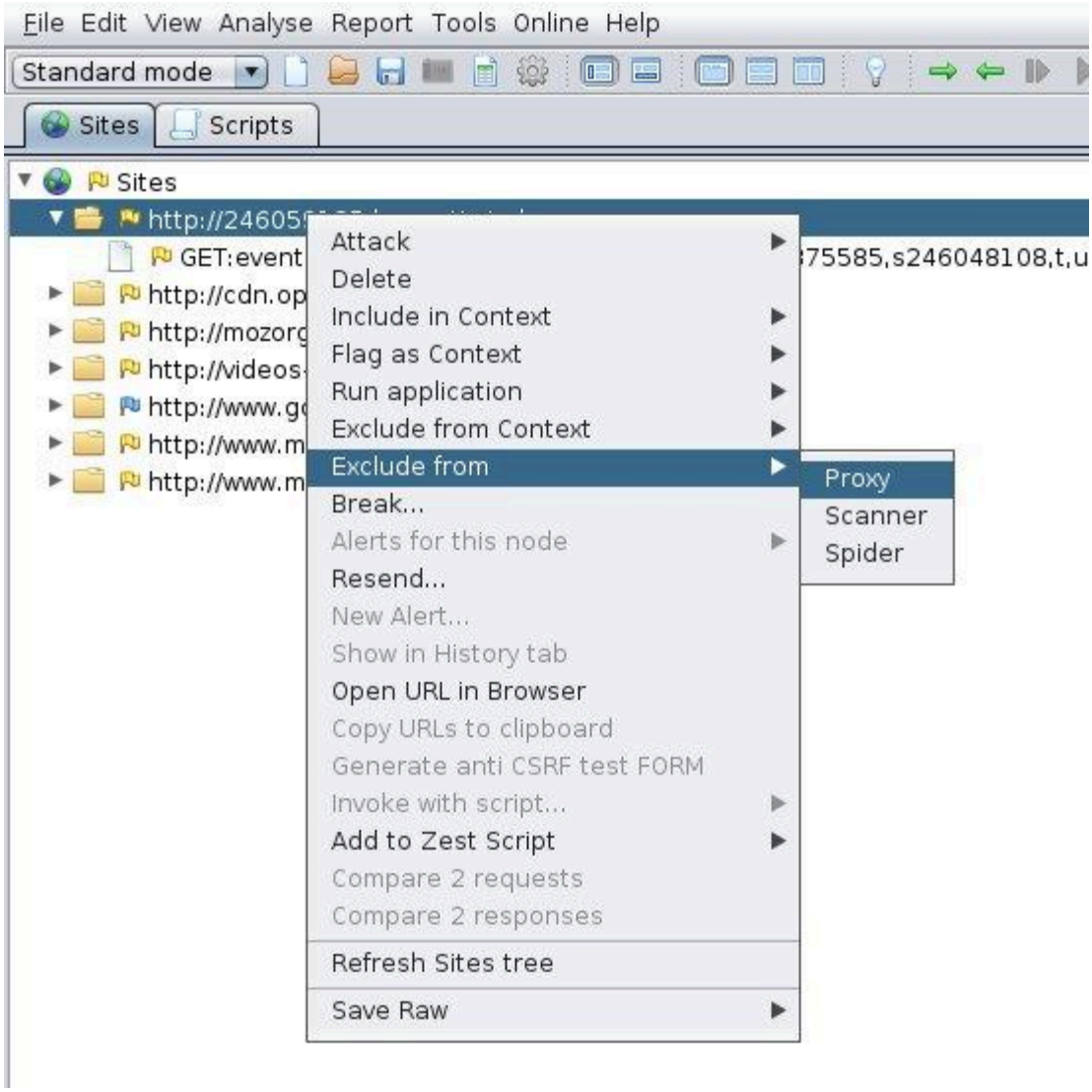
End

Configuring the URL Filter

Complete this procedure to use the GUI to configure URLs that the proxy should ignore.

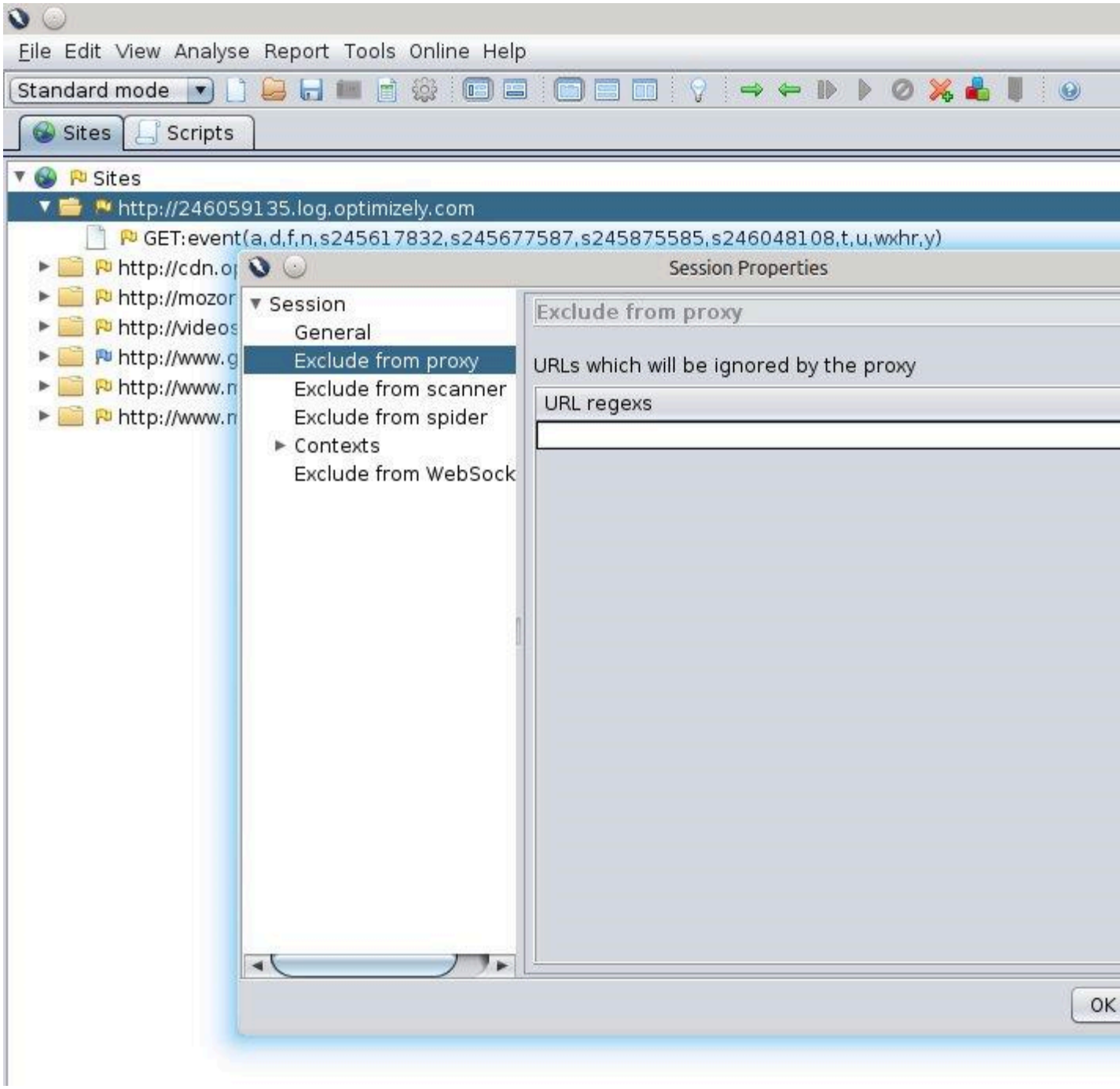
Start You can exclude a site in one of two ways:

- In the **Sites** tab, right-click on a site and select **Exclude from > Proxy**.



Select a site to exclude

- Select **File > Properties**. In the Session Properties window, select **Exclude from proxy**, add your URL, and click **OK**.



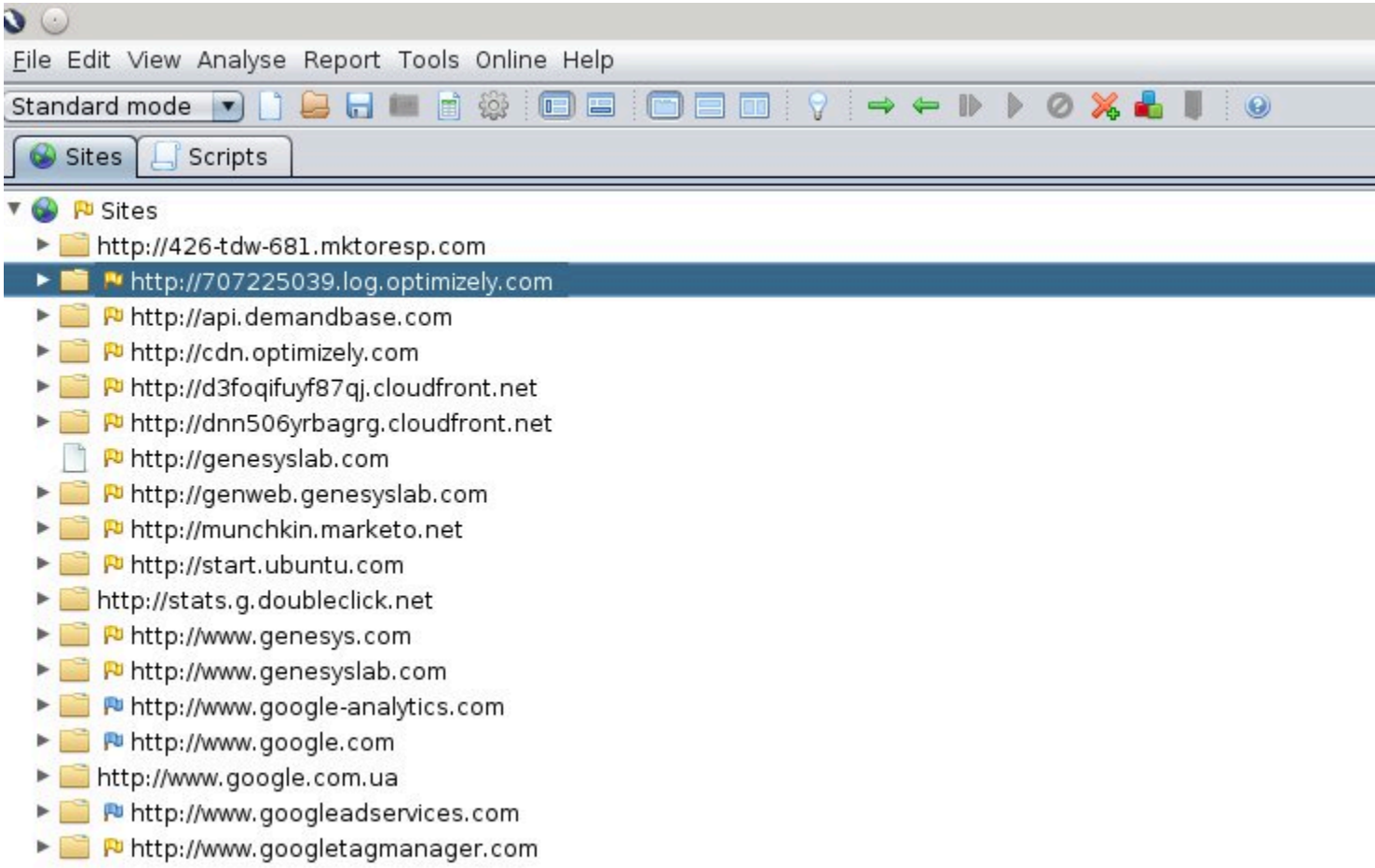
Enter a URL to exclude.

- If you want the proxy to remember the excluded URLs beyond the current session, select **File > Persist session...** and select a file to save your session.

End

Working with the Proxy

After you have configured the proxy, keep it open and open up a web browser. Now you can browse through your web pages that are instrumented with Genesys Web Engagement and they will be displayed in the **Sites** tab of the proxy GUI:



Your instrumented pages show up in the **Sites** tab

For more information about working with ZAProxy, see https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

Security Testing with ZAProxy

Genesys performs security testing with OWASP Zed Attack Proxy (ZAProxy) to make sure the Genesys Web Engagement solution is invincible to known attacks.

ZAP Overview

The ZAProxy is an easy-to-use, integrated penetration testing tool for finding vulnerabilities in websites and web applications.

Among others, ZAProxy supports the follow methods for penetration security testing:

- passive scan
- active scan

Genesys uses both methods.

Passive Scan Overview

ZAP is an Intercepting Proxy. It allows you to see all of the requests made to a website/web app and all of the responses received from it. For example, you can see AJAX calls that might not otherwise be obvious.

Once set up, ZAP automatically passively scans all of the requests to and responses from the web application being tested.

While mandatory use cases for the application that is being tested are followed (either manually or automatically), ZAProxy analyzes the requests to verify the usual operations are safe.

Active Scan Overview

Active scanning attempts to find potential vulnerabilities by using known web attacks against the selected targets. Active scanning is an attack on those targets. ZAProxy emulates known attacks when active mode is used.

Through active scanning, Genesys Web Engagement is verified against the following types of attacks:

- **Spider attack** — Automatically discovers all URL links found on a web resource, sends requests, and analyzes results (including src attributes, comments, low-level information disclosure, and so on).
- **Brute browsing** (based on the Brute Force technique) — Systematically makes requests to find secure resources based on known (commonly used) rules. For example, backup, configuration files, temporary directories, and so on.
- **Active scan** — Attempts to perform a predefined set of attacks on all resources available for the web resource. You can find the default set of rules here.
- **Ajax spider** — Automatically discovers web resources based on presumed rules of AJAX control (JS scripts investigation, page events, common rules, dynamic DOM, and so on).

> **Important**
>
> Requests to other web applications must be excluded from scanning in order to see a report for a particular web application.

> **Important**
>
> Web applications that are being tested should be started on the local box because some types of verification (like active scanning) can be forbidden by network administrators.

References

If you want to examine your website against vulnerabilities in a similar way, refer to the OWASP Zed Attack Proxy Project or other documentation to learn about how to perform security testing with ZAP.

Sample Applications

Genesys Web Engagement includes a sample called **Playground**.

The Playground environment also includes a website, and you can use this site with the Playground application to find out what Web Engagement can do.

The Web Engagement Quick Start Guide provides step-by-step instructions on how to use Playground to see Web Engagement in action.

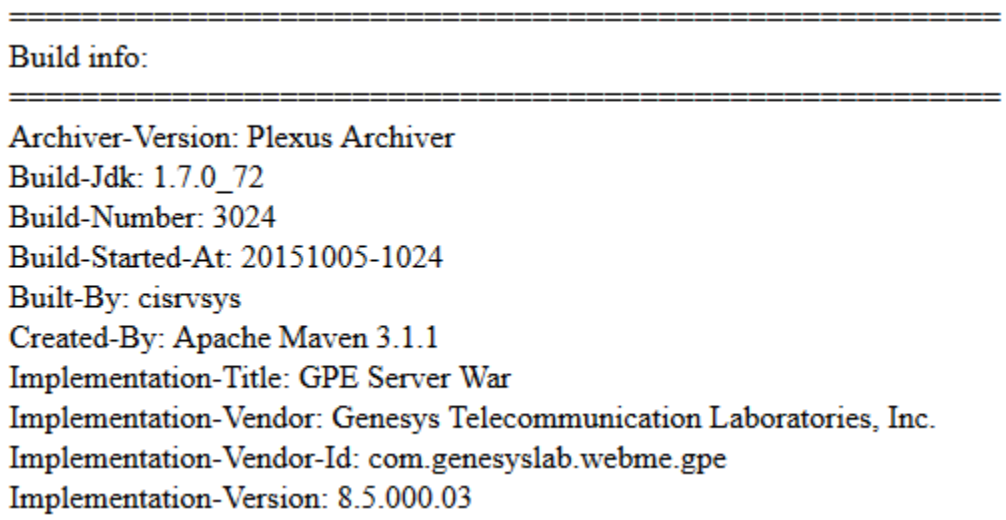
Get Information About Your Application

After you have started your Web Engagement Servers, you can explicitly request version information from them.

To get this information, you should send a GET HTTP request to the appropriate URL for the server.

URL: `http(s)://<web_engagement_server_host>:< web_engagement_server_port>/server/about`

Note: This image shows an example from Web Engagement Server 8.5.0. The information on your screen might differ from what is shown here.



An example Web Engagement Server response

Media Integration

> **Important**
> This article is only for use with native Web Engagement widgets. If you plan to use Genesys Widgets, you must follow these customization instructions.

You can integrate Genesys Web Engagement with second-party and third-party media to extend its capabilities beyond what is available with the basic GWE installation. The key integration points for both media types are the Notification Service or proactive invitation:

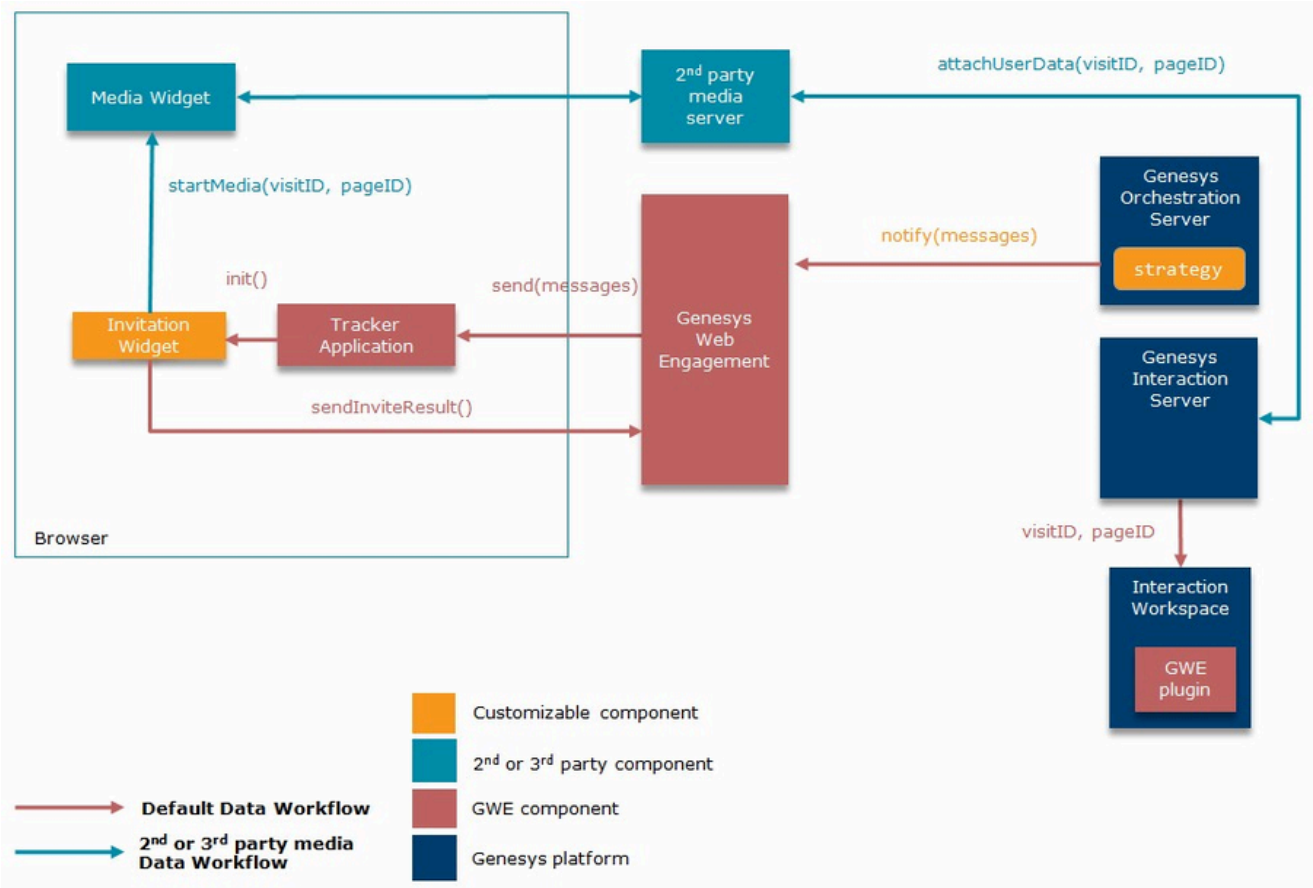
- **The second-party media** is a first-class citizen in the Genesys platform that can carry extra business attributes (attached data), like **visitID**, **pageID**, and so forth, for operational and reporting purposes. The key differentiator is that the second-party media is processed by Genesys components like Interaction Server. The principle of the integration is simple — taking control of the proactive invitation and Notification Service. Examples of second-party media include GWE Chat, Genesys Mobile Services (GMS) Chat, and Web API Chat.
- **The third-party media** is provided by third-party services that are not tightly integrated with the Genesys cross-channel platform (particularly with Interaction Server). The integration with third-party media boils down to taking control of the proactive invitation, which is part of the Notification Service.

The **proactive invitation** (represented by the Invitation Widget) is the key integration point that should be used when you need to overlay the widget on a page. The **Notification Service** should be used in all other cases.

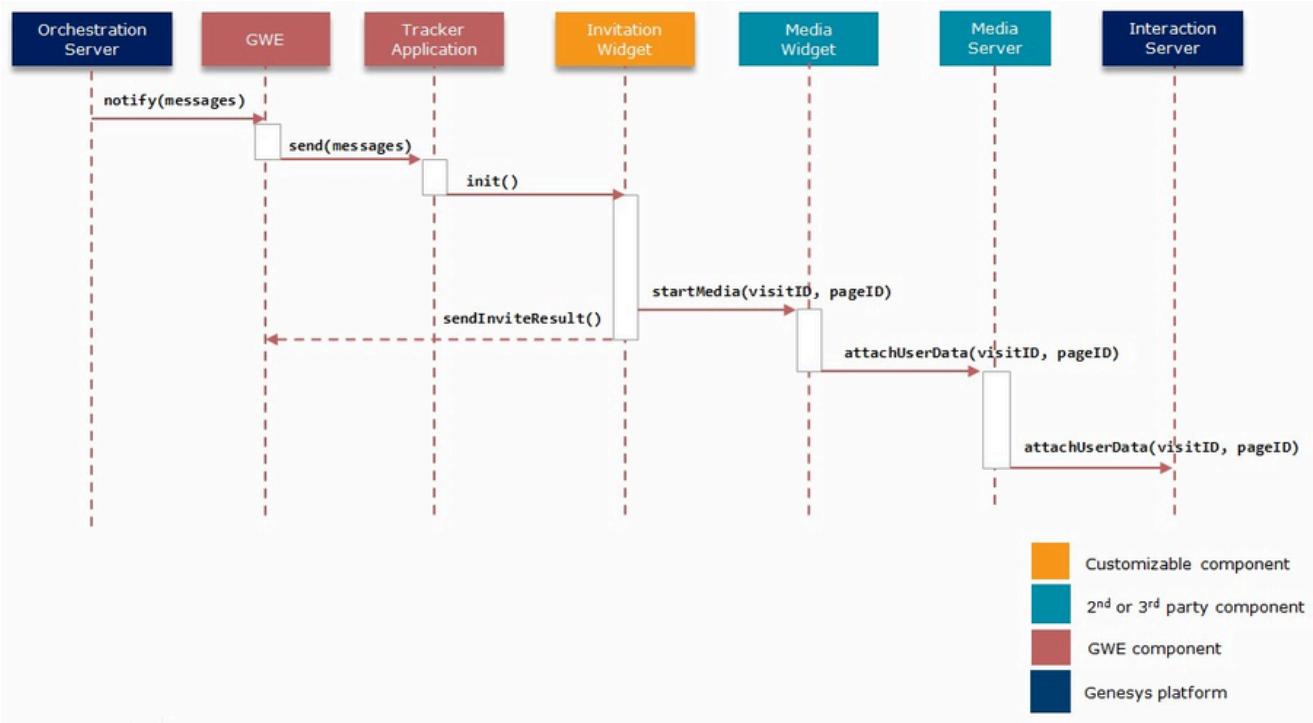
Integration with Genesys Widgets

In order to integrate with Genesys Widgets, the media widget and media server components must propagate the Web Engagement **visitID** and **pageID** attributes to the interaction as attached data. You can get the **visitID** and **pageID** in the widget through the public **_gt.push** ['getIDs',callback] method in the Monitoring JS API. For proactively created chat sessions, you must attach a key-value pair with a key of **webengagement** and an empty string as the value. This key-value pair can be used later to distinguish between chat sessions that have been created proactively and reactively.

The diagram below shows an example of the data flow between components in a second-party media integration. Web engagement is initiated by Genesys Orchestration Server (ORS), which sends a notification to Genesys Web Engagement. As a result, the custom Invitation Widget appears in the browser. After the invitation is accepted by the user, the Invitation Widget passes the Web Engagement attributes (**visitID** and **pageID**) to the Media Widget. The third-party media server then starts a new interaction with the attributes as attached data. Based on this data, the Web Engagement Plug-in for Interaction Workspace can provide the browser history of the current user and other information.

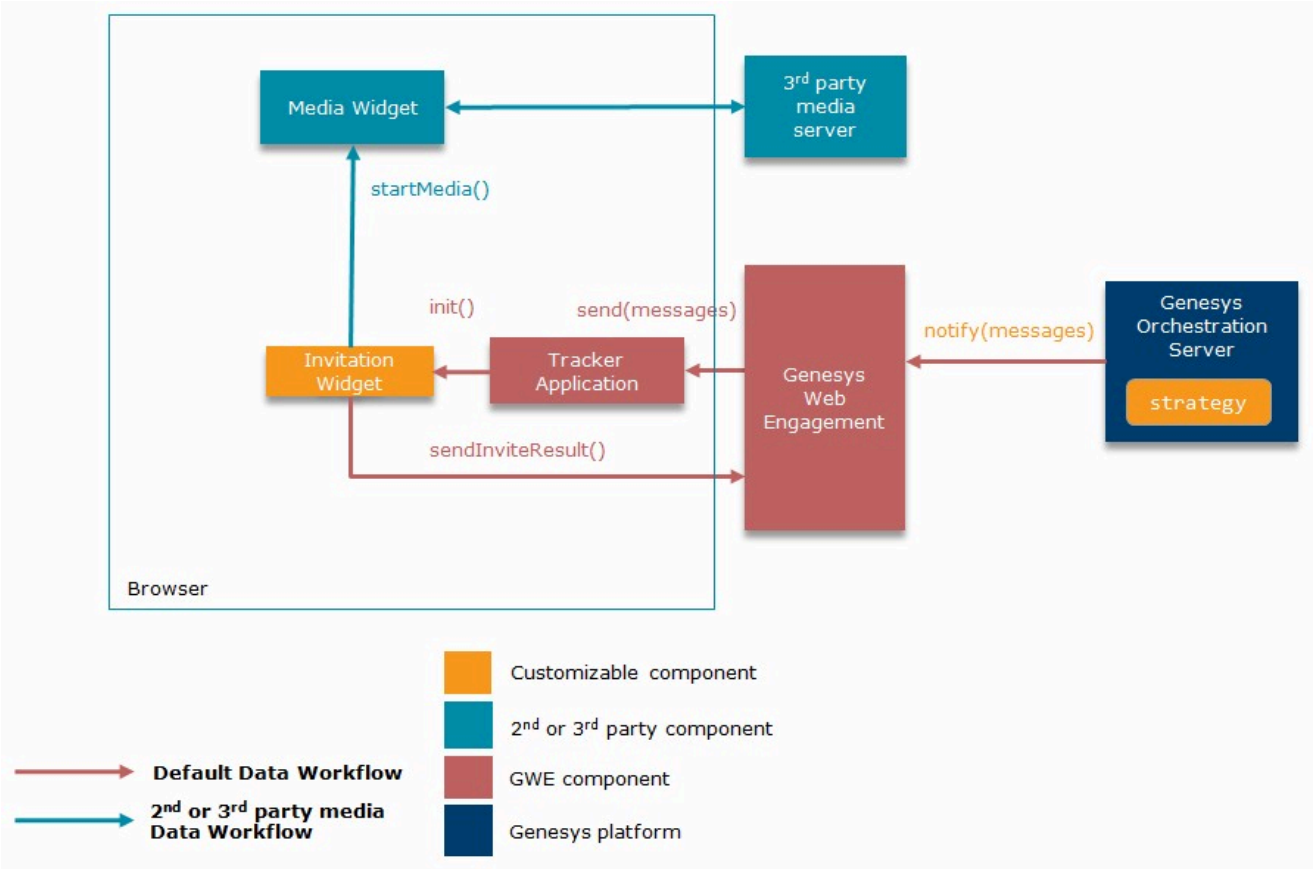


Here is another view of the data flow in a second-party media integration, shown in a sequence diagram:



Third-Party Media Integration

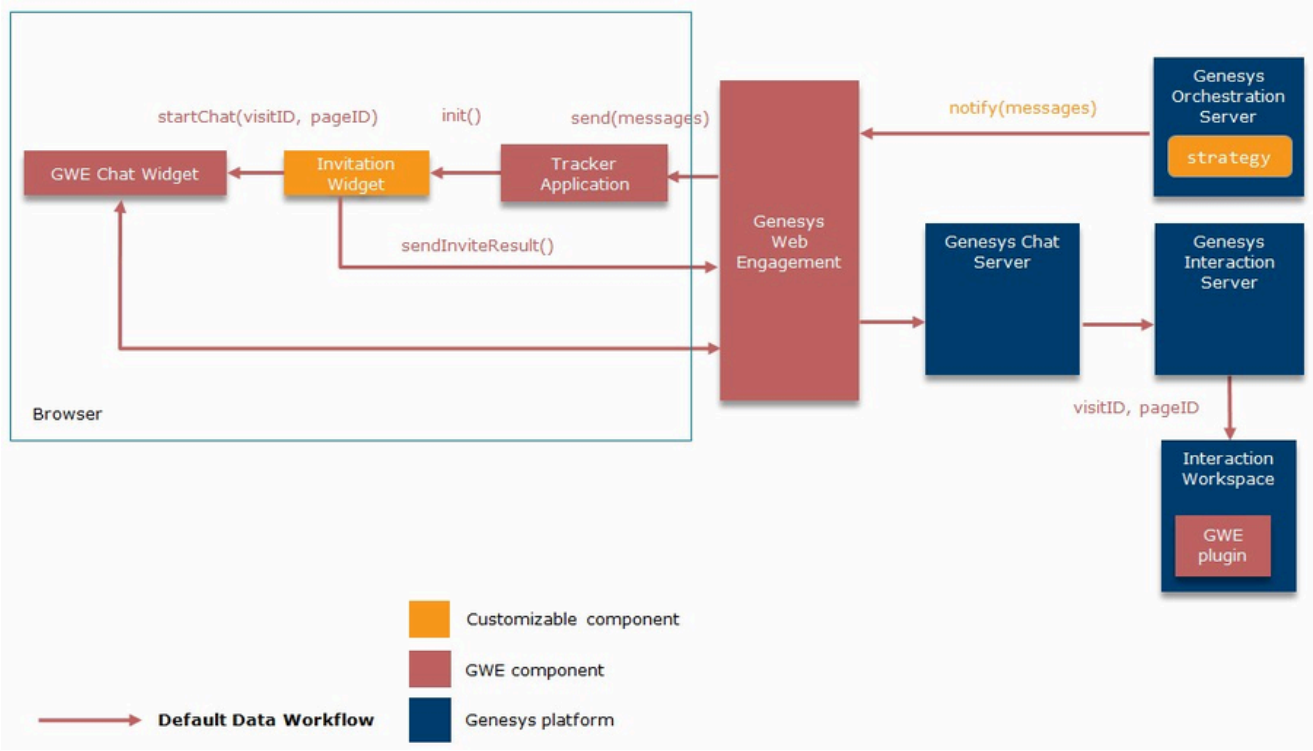
The diagram below shows an example of third-party media integration. Web engagement is initiated by ORS, which sends a notification to Genesys Web Engagement by using the Notification Service REST API. As a result, the custom Invitation Widget appears in the browser. After the invitation is accepted by the user, the Invitation Widget initiates the Media Widget. The third-party media server does not create an interaction in Genesys Interaction Server as it does in the second-party media integration scenario, but the same customization points are still available: Notification Service and proactive invitation.



Examples

GWE Chat Integration

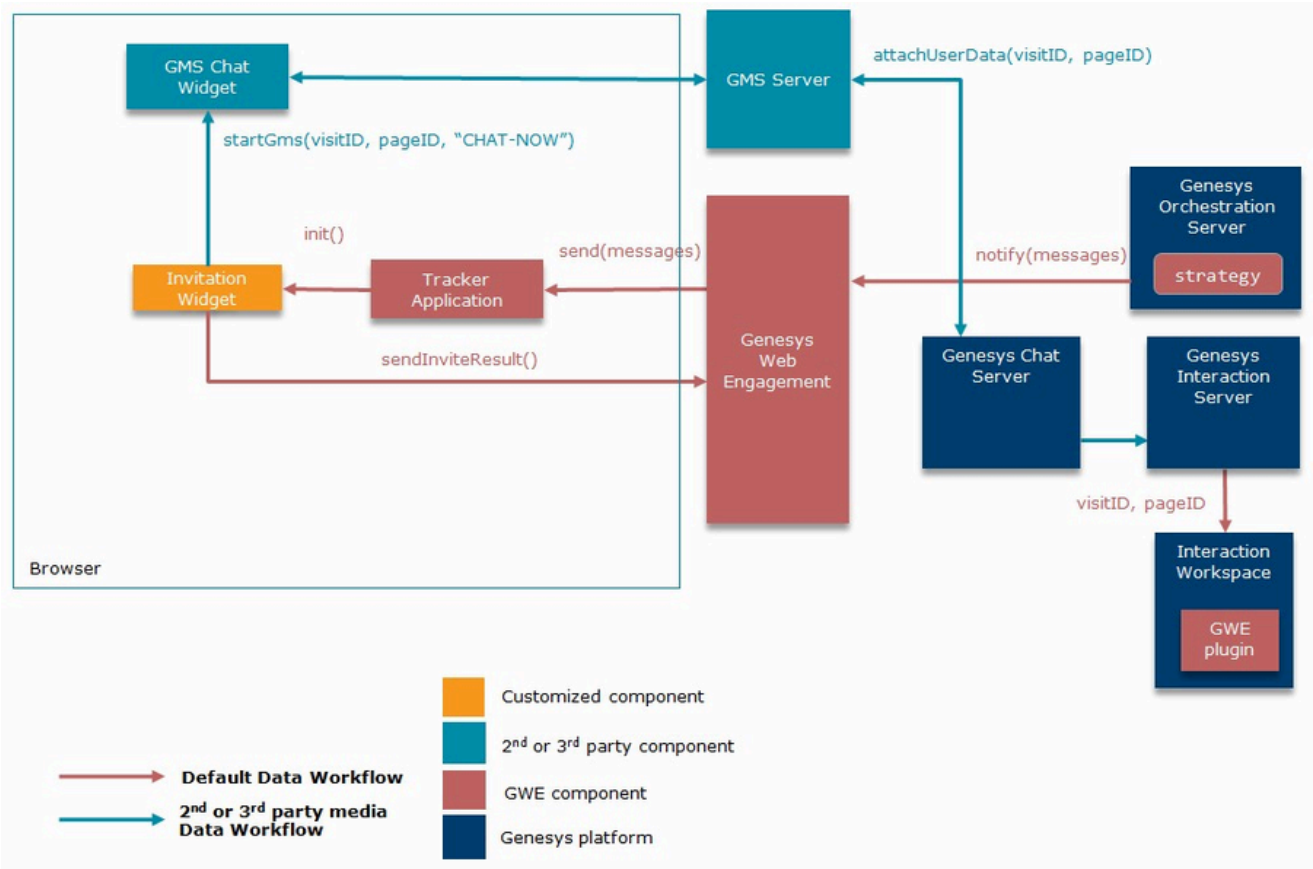
Genesys Web Engagement chat use the same integration path as described in the Second-Party Media Integration section:



GMS Chat Integration

Let's look at how to integrate the second-party chat offered by Genesys Mobile Services instead of the standard Genesys Web Engagement chat. In this example, we use the GMS Chat Widget and initiate a chat session when the user accepts the proactive invitation.

The diagram below shows the data flow between components involved in the integration:



To integrate GMS with Genesys Web Engagement, we need to modify the following:

- GWE Proactive Invitation
- GWE Engagement Logic Strategy
- GMS Chat Widget

GWE Proactive Invitation

The proactive invitation is represented by the **invite.html** file (see Invitation Widget for details), but Genesys recommends that you make a copy of this file to modify for the integration. In this example, we use a copy called **inviteGMS.html**.

In this file, we need to change how the invitation reacts when it is accepted by a visitor. We can do this in the `onAccept()` function, which checks the invitation type and calls either `startChat()`. Since we want to integrate chat, we need to replace the standard `startChat()` with our own function called `startGms()`. This function opens the GMS Chat Widget window (**indexGPE.html** — we will create this file in the GMS Chat Widget section below) and passes the `gmsScenario` variable.

```
...
function startGms(gmsScenario) {
    openWindow(
        'http://<GMS Host>/genesys/admin/js/sample/cb/indexGPE.html',     // Customized GMS
widget
```

```
        'GMS',                                                          // Window title
        gmsScenario                                                     // GMS scenario name
    );
}
function onAccept() {
    log('onAccept()');
    closeInviteDialogWindow();
    if (_config.type === INVITE_TYPE.CHAT) {
        startGms('CHAT-NOW');  // Start GMS 'CHAT-NOW' scenario
        sendInviteResult(INVITE_RESULT.ACCEPT_CHAT);
    } else if (_config.type === INVITE_TYPE.CALLBACK) {
        startCallback();
        sendInviteResult(INVITE_RESULT.ACCEPT_CALLBACK);
    } else {
        error('Invitation type not defined');
    }
}

...
```

> **Important**
>
> You can add callback integration the same way. Replace the `startCallback()` function with your own appropriate function in the `onAccept()` handler.

GWE Engagement Logic Strategy

In the previous section we made a new invitation widget for GMS chat, called **inviteGMS.html**, and now we need to modify the Engagement Logic Strategy to use this widget. The final notification message should look like the following:

```
...
var notification_message = [ {
        'page': event.pageID,
        'channel': 'gpe.appendContent',
        'data': {
            'url': '/server/api/resources/v1/inviteGMS.html'
        }
    } ];
...
```

> **Important**
>
> For more information about Engagement Logic, see Start Engagement as a Result of the Engagement Logic Strategy.

GMS Chat Widget

The GMS Chat Widget is represented by the **index.html** file, which is included as part of the GMS Javascript (Web) Sample. Again, Genesys recommends that you make a copy of this file to modify for the integration. In this example, we use a copy called **indexGPE.html**.

The GMS Chat Widget is an HTML page that can be loaded as either an iframe or a pop-up, which makes it simple to pass additional data through URL variables. In the GWE Proactive Invitation section, we added the `gmsScenario` variable to the URL in the `startGms()` function. Now we need to change the GMS Chat Widget so that it automatically starts the GMS scenario defined in that variable.

First, we need to get `gmsScenario` from the URL:

```
...
    function getUrlVars (name) {
        var vars = [], hash, i,
                hashes = window.location.href.slice(window.location.href.indexOf('?') +
1).split('&');
        for (i = 0; i < hashes.length; i += 1) {
            hash = hashes[i].split('=');
            vars.push(hash[0]);
            vars[hash[0]] = hash[1];
        }
        return vars[name];
    }
...
```

Next, we need to change the scenario name and connect to GMS Server:

```
...
    function gpeStartScenario() {
        var scenario = getUrlVars('gmsScenario') || 'CHAT-NOW';     // Fetch scenario name.
Default is 'CHAT-NOW'
        $('#settings [name=service_name]').val('samples_new');      // Example GMS Service
        $('#scenario').val(scenario);                               // Set scenario name

        connect();                                                  // Connect to GMS
    }
...
```

Finally, we need to add the required parameters (**visitID** and **pageID**) to the `connect()` function, which is responsible for setting up the connection to GMS Server:

```
...
    function connect(e) {
        // get data from ui
        var headers = new Object();
        headers.gms_user = $('#user_name').val();
        var params = new Object();
        params.first_name = $('#first_name').val();
        params.last_name = $('#last_name').val();
        params._provide_code = $('#provide_code').val();

        params.visitID = getUrlVars('visitID');             // Required parameters
        params.pageID = getUrlVars('pageID');               // Required parameters

        var scenario = $('#scenario').val();
        if ($('#scenario').val() == "VOICE-SCHEDULED-USERTERM") {
            params._desired_time = $('#available_time_slots').val();
        }
        var serviceName = $('#service_name').val();
        var serviceUrl;
        var responseHandler = onResponseReceived;
        if (scenario == "REQUEST-INTERACTION") {
            serviceUrl = 'request-interaction';
            // request interaction requires _phone_number instead of _customer_number as
required by callback
```

```
            params._phone_number = $('#contact_number').val();
            responseHandler = onBuiltinCallbackResponseReceived;
        } else if (scenario == "REQUEST-CHAT") {
            serviceUrl = 'request-chat';
            params._customer_number = $('#contact_number').val();
            responseHandler = onBuiltinCallbackResponseReceived;
        } else {
            serviceUrl = 'callback/' + serviceName;
            params._customer_number = $('#contact_number').val();
        }
        // post data
        gmsInterface.createCallback(scenario, $('#url').val(), serviceUrl, params, headers,
responseHandler);
        //gmsInterface.call_agent();
    }
...
```

Now that we've customized the GMS Widget, it can be started automatically with a connection to GMS Server in `gpeStartScenario()`.

```
// inside onready callback
gpeStartScenario();
```

Using Pacing Information to Serve Reactive Requests

General information about Pacing Algorithms

The Web Engagement pacing component is designed to predict the number of media interactions that should be proactively generated by the Web Engagement Server in each succeeding time interval. For more information about pacing, consult this article.

Web Engagement also supports dual pacing, in which the pacing algorithm is able to determine how much of its capacity should be set aside in order to handle reactive traffic without allowing the proactive traffic to exceed the desired range.

In order to work with dual pacing, you should understand that:

- The pacing component works with a set of Agent Groups.
- The term *Channel* refers to a set of Agent Groups in which each group of agents is configured to work on the same, specific media channel, such as chat or Web RTC.
- The pacing component makes predictions for each Agent Group separately by creating a dedicated thread for each Agent Group and running an instance of the pacing algorithm in each one.
- The pacing algorithm is executed at the frequency specified by the refreshPeriod option in the [pacing] section.
- The pacing algorithms used for each Agent Group monitored by the pacing component are identically configured.
- In addition to group-based predictions, the pacing component also calculates consolidated results for every channel—that is, the sum of the results for all groups belonging to a particular channel.
- There are two types of workflows:
  - **Proactive**—in which a media interaction is created every time a visitor accepts a proactively generated invitation (that is, an invitation that was triggered by specific rules associated with the Web Engagement software). With a proactive workflow, Web Engagement has complete control over when and if a given interaction is created.
  - **Reactive**—in which media interactions are created as a result of a visitor's reaction to static elements on the website, such as clicking a button or following a link. This kind of workflow is beyond the control of the Web Engagement software, since it can't control the behavior of the people who visit the site.

  Note that both proactive and reactive workflows produce the same kinds of media interactions, such as chat interactions. But from the standpoint of the pacing component, proactively and reactively generated interactions have vastly different implications.

- When the pacing component is configured to calculate information for both proactive and reactive

workflows, we say that it is in *dual mode* and that it has been configured to use a *dual pacing algorithm*.

- **Proactive** workflow predictions can be calculated in *both* the simple proactive mode *and* in dual mode. But **Reactive** predictions are *only* calculated in dual mode.
- The pacing component cannot distinguish between Agent Groups that have been configured to service proactive workflows and ones that are servicing reactive workflows. This distinction is completely controlled by your Genesys configuration, including the way your strategies are configured.
- The pacing component assumes that each agent it is monitoring only belongs to one of the Agent Groups it is monitoring.
- You can set up an environment where an Agent Group is configured to work with several interaction types (or channels) simultaneously. This is known as *blended* mode. In blended mode, the pacing component executes a dedicated instance of the pacing algorithm for each channel that is configured for a particular Agent Group.
- The pacing algorithms use statistical information obtained both from Stat Server and from the Web Engagement software, which has access to information that can't be obtained from Stat Server, such as the pending invitation count and the average time it takes to obtain a disposition code for an invitation.

Configuring dual pacing mode

You can specify which type of pacing algorithm to use by setting the algorithm option in the [pacing] section. This option supports the following values:

- **SUPER_PROGRESSIVE**—The Super Progressive optimization method only affects the Abandonment Rate parameter and provides a higher Busy Factor then the Predictive one. It is efficient for relatively small agent groups (1 to 30 agents) when the Predictive method gives poor results.
- **PREDICTIVE_B**—A Predictive method based on the Erlang-B queuing model. Recommended for large agent groups (more than 30 agents) with impatient customers who cannot stay in the queue, even for a short time.
- **SUPER_PROGRESSIVE_DUAL**—An adaptation of the Super Progressive method for environments serving both proactive and reactive interactions.
- **PREDICTIVE_B_DUAL**—An adaptation of the Predictive B method for environments serving both proactive and reactive interactions.

As you can see, you must specify either **SUPER_PROGRESSIVE_DUAL** or **PREDICTIVE_B_DUAL** if you want to use a dual pacing algorithm.

The most important parameter calculated by a simple pacing algorithm is called **InteractionsToSend**. This parameter determines how many proactive invitations should be sent during each refresh period. When you use a dual pacing algorithm, you need to set a balance between the percentage of agents in each group who are handling proactive invitations and those who are handling reactive ones. Without doing this, you run the risk of having your reactive traffic take over, meaning that proactively created hot leads—people who are likely to be prime customers—may be displaced by random visitors about whom you know nothing.

You can use the proactiveRatio option to adjust this balance.

Web Engagement helps avoid this issue by calculating the **InboundPortion** parameter, which specifies how much capacity should be set aside for inbound (reactive) traffic. The calculated values

for **InboundPortion** can range from 0 to 1:

- 0 means that the affected page should not allow inbound traffic (for example, by disabling chat request buttons). This value will be returned by the pacing algorithm in situations where each new reactive chat request "seizes" an agent who could potentially handle a proactive chat session, thereby making it impossible to serve proactive traffic.
- 1 means that there are enough agents to serve the predicted count of proactive invitations, even if reactive interactions are started on the affected page.
- A value between 0 and 1 means that if a reactive interaction is started on the affected page, then it can potentially seize an agent who would otherwise be serving a predicted proactive interaction. This situation may be undesirable, especially if the potential value of your proactive interactions is high. In that case, you probably want to suppress the calculation of **InboundPortion**.

Suppressing Calculation of InboundPortion

Web Engagement provides two ways to suppress the calculation of **InboundPortion**:

- Use a simple, proactive-only pacing algorithm. In this case, **InboundPortion** will not be calculated at all.
- Use a dual pacing algorithm, but specify proactiveRatio at 100. In this case, the value of **InboundPortion** will always be 0, meaning that the affected page is instructed to block all inbound chat traffic, if possible—for example, by disabling chat request buttons.

The Pacing REST API

For times when reactive chats can only be controlled from the page, Web Engagement provides a RESTful Pacing API that gives you access to the value of the **InboundPortion** parameter calculated by the dual pacing algorithm.

You can also use the Pacing API to access statistical information about agent availability in the monitored Agent Groups. Although this statistical information is provided in a raw format that is used as input by the pacing algorithm, it can sometimes be critical for your understanding of how to control activity from the affected page.

Obtaining the Reactive State

Reactive state is another term that is used when talking about the **InboundPortion** parameter described above.

You can query the reactive state by issuing this request:

```
http://<gweserver.host:gweserver.port>/server/data/pacing/
reactiveState?channel=<channelName>&groups=[<names>]
```

The information returned by this request helps you understand whether reactive traffic is displacing proactive traffic on the specified channel for the specified Agent Group. If an Agent Group is not specified, the result will be calculated for the entire channel.

The response to this request is a float between 0 and 1 that indicates the probability with which the

affected page should allow reactive interaction:

- **1**—There are no limitations on the number of reactive interactions.
- **0**—The page should not allow any reactive interactions.
- If the value is between `0` and `1`, the page should use the specified probability to determine whether to allow a given reactive interaction.

Let's consider an example of this last situation. If the **reactiveState** request returns a value of `0.7`, this means that you probably only want 7 out of 10 of your recently loaded pages to allow reactive interactions. Therefore, the other 3 pages should prohibit them. If you don't set up this kind of scenario, newly created reactive interactions can spiral out of control, meaning that some of them will seize agents who should have been left available for proactive customers. This means that Web Engagement will produce failed hot leads.

In JavaScript you can issue a **reactiveState** call like this:

```
<script>
    $.ajax({url: 'http://{server}:{port}/server/data/pacing/reactiveState?channel=chat'})
    .done(function( result ) {
        console.log('result: ' + result.reactiveState);
        var rndValue = Math.random();
        if(rndValue > result.reactiveState) {
            // Disable reactive chat buttons
        }
        else {
            // Enable reactive chat buttons
        }
    });
</script>
```

Here's a sample:

```
http://example.com:9081/server/data/pacing/
reactiveState?channel=chat&groups=Web%20Engagement%20Chat
```

And the response:

```
{"reactiveState":1.0}
```

Note: This example uses the jQuery JavaScript library, which requires that jQuery be loaded on the page.

Obtaining Channel Capacity

You can use the **channelCapacity** method to understand how many concurrent interactions to allow on a specific channel for a specific Agent Group (or for the specified channel only, if a group is not explicitly specified).

Important: This method takes into account both agent state and the capacity rules that have been configured for each agent. For example, if the channel contains 1 Ready agent with a capacity of 2 and 1 Ready agent with a capacity of 3, then the cumulative channel capacity will be calculated as 5.

Important: An **InboundPortion** value of 1 does not always mean that a reactive chat will be immediately delivered to an agent.

Let's consider a situation where no agents are ready in the system and the proactive traffic is predicted at 0. This means that the value of **InboundPortion** will be 1 (because there isn't any proactive traffic to displace). However, because none of our agents are ready, you also don't want to allow any immediate reactive interactions.

By issuing a channel capacity request, you can get more information on whether or not you have to allow new reactive interactions.

Here's how to call the method:

```
http://<gweserver.host:gweserver.port>/server/data/pacing/
channelCapacity?channel=<channelName>&groups=[<names>]
```

And here is an example of how to use it in a script:

```
<script>
    $.ajax({url: 'http://{server}:{port}/server/data/pacing/channelCapacity?channel=chat'})
    .done(function( result ) {
        console.log('Chat channel capacity is: ' + result.capacity);
    });
</script>
```

This request:

```
http://example.com:9081/server/data/pacing/
channelCapacity?channel=chat&groups=Web%20Engagement%20Chat
```

Might yield this response:

```
{"capacity":254}
```

Note: The channel capacity request provides information about the current state of channel. But you need to keep in mind the potential for race conditions.

For example, if ten browsers have requested the channel capacity concurrently, each of them could be told that the value is 1. By itself, this would lead each browser session to think that it can trigger a reactive interaction. But if an interaction is triggered on more than one browser, you will have a race condition in which the first interaction to seize an agent will use up all of the available capacity, and all other interactions will be in a wait state.

Note: This example uses the jQuery JavaScript library, which requires that jQuery be loaded on the page.

Step-by-Step Examples

Let's consider an example of how to use pacing information to determine how to serve reactive chats.

There are 2 use cases:

- The page makes sure that proactive traffic is not displaced.
- The page is not aware of proactive traffic and is interested only whether any agents are Ready.

Making sure that proactive traffic is not displaced

This is the most general use case, in which you need to avoid two different pitfalls:

- Reactive interactions should not be allowed to displace potential proactive interactions (which are calculated based on the result of the **reactiveState** method)
- Reactive interactions should only be triggered when at least one Ready agent is available on the channel

Here is the algorithm for this situation:

1. Determine whether reactive interactions are undesirable. If so, disable the request buttons on the page.
2. If reactive interactions are allowable, find out whether there are any available agents.
3. If no agents are available, disable the request buttons on the page.
4. If one or more agents are available, make sure the request buttons are enabled.

And here is a JavaScript sample:

```
function reactiveChatPacing() {
    $.ajax({url: 'http://{server}:{port}/server/data/pacing/reactiveState?channel=chat'})
        .done(function (reactiveResult) {
            var rndValue = Math.random();

            // Check that reactive chat is allowed with probability result.reactiveState
            if (rndValue >= reactiveResult.reactiveState) {
                disableReactiveChatButtons();
            } else {

                // For the case result.reactiveState == 1 we should check channel capacity
                // as there is no guarantee that there are Ready agents
                if (reactiveResult.reactiveState == 1) {

                    $.ajax({url: 'http://{server}:{port}/server/data/pacing/
channelCapacity?channel=chat'})
                        .done(function( capacityResult ) {
                            if (capacityResult.capacity == 0) {
                                disableReactiveChatButtons();
                            } else {
                                enableReactiveChatButtons();
                            }
                        });

                }
                else {
                    enableReactiveChatButtons();
                }
            }
        });
}


function disableReactiveChatButtons () {
    // Disable reactive chat buttons
}
```

```
function enableReactiveChatButtons() {
    // Enable reactive chat buttons
}
```

Note: This example uses the jQuery JavaScript library, which requires that jQuery be loaded on the page.

Ignoring proactive traffic

This case is a shorter variant of the first one, since you only need to determine the channel capacity.

Note that you should reserve the use of this approach for situations in which you only want to support reactive interactions.

Here is the algorithm:

1. Find out whether any agents are available.

And the JavaScript:

```
function reactiveChatChannelCapacity() {
    $.ajax({url: 'http://{server}:{port}/server/data/pacing/channelCapacity?channel=chat'})
        .done(function (capacityResult) {
            if (capacityResult.capacity == 0) {
                disableReactiveChatButtons();
            } else {
                enableReactiveChatButtons();
            }
         });
}

function disableReactiveChatButtons () {
    // Disable reactive chat buttons
}

function enableReactiveChatButtons() {
    // Enable reactive chat buttons
}
```

Note: This example uses the jQuery JavaScript library, which requires that jQuery be loaded on the page.

Some Sample Calculations

70% Proactive Traffic, 30% Reactive Traffic

1. First, set your configuration options like this:
    - refreshPeriod = 2 (default value)

- proactiveRatio = 70
- optimizationGoal = 3 (default value)
- optimizationTarget = ABANDONMENT_RATE (default value)
- algorithm = SUPER_PROGRESSIVE_DUAL

2. Then get the **InboundPortion** value by using the corresponding HTTP request on the browser side.
3. If **InboundPortion** is 1, check the channel capacity.
4. Either reduce or increase the reactive traffic, or leave it alone—depending on the result of your request, as shown in the above example script.

30% Proactive Traffic, 70% Reactive Traffic

1. First, set your configuration options like this:
   - refreshPeriod = 2 (default value)
   - proactiveRatio = 30
   - optimizationGoal = 3 (default value)
   - optimizationTarget = ABANDONMENT_RATE (default value)
   - algorithm = PREDICTIVE_B_DUAL
2. Then get the **InboundPortion** value by using the corresponding HTTP request on the browser side.
3. If **InboundPortion** is 1, check the channel capacity.
4. Either reduce or increase the reactive traffic, or leave it alone—depending on the result of your request, as shown in the above example script.

Disable Reactive Traffic

That is, provide 100% proactive traffic by disabling all reactive chats.

1. First, set your configuration options like this:
   - refreshPeriod = 2 (default value)
   - proactiveRatio = 100
   - optimizationGoal = 3 (default value)
   - optimizationTarget = ABANDONMENT_RATE (default value)
   - algorithm = SUPER_PROGRESSIVE_DUAL
2. Then get the **InboundPortion** value by using the corresponding HTTP request on the browser side (it must be 0).
3. Deny reactive traffic by disabling your chat buttons.

Disable Proactive Traffic

Provide 100% reactive traffic.

1. First, set your configuration options like this:
   - refreshPeriod = 2 (default value)
   - proactiveRatio = 0
   - optimizationGoal = 3 (default value)
   - optimizationTarget = ABANDONMENT_RATE (default value)
   - algorithm = SUPER_PROGRESSIVE_DUAL
2. Then get the **InboundPortion** value by using the corresponding HTTP request on the browser side (it must be 100).
3. Allow reactive traffic by enabling your chat buttons.

Dynamic Multi-language Localization Application Sample

Prerequisites

- Use the latest version of Genesys components.

Creating multilingual categories

Create one or more categories by following the instructions in Creating a Category.

> **Important**
> All tags for multi-language categories must have a different expression.

Dynamically adding the language in the instrumentation script

The language code is transmitted as a URL parameter. You can pass a language code as part of the URL or you can set the code statically.

Here is an example of the code as part of the URL:

```
http://<Web Engagement Server host>:<Web Engagement Server port>/multi/
main.jsp?title=◈◈&language=zh-CN
```

Placing the language code in the instrumentation script allows you to localize the registration form and chat. To do this, complete the following:

1. Add Localization Files to Your Web Engagement Application.
2. Add the language code to your instrumentation script.

The following example shows how to add the language code to your instrumentation script:

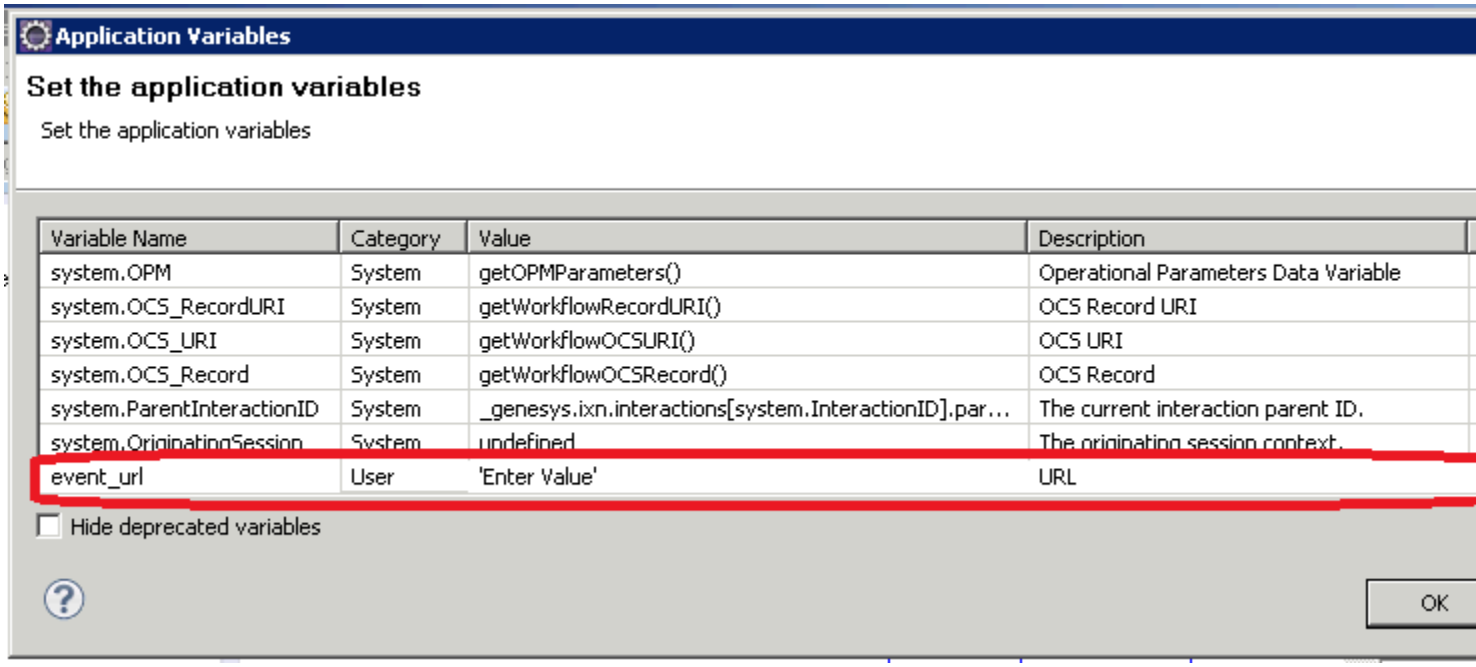
```
<% String title = request.getParameter("title"); %>
<% String langCode = request.getParameter("language"); %>
<title><%=title%></title>
<script>
```

```
  var _gt = _gt || [];
  _gt.push(["config", {
    "name" : "multi",
    "domainName" : "<domain name of your website>",
    "server" : "432",
    "languageCode" : "<%=langCode%>",
    "dslResource" : "<Web Engagement Server host>:<Web Engagement Server port>/server/api/
resources/v1/dsl/domain-model.xml",
    "secureDslResource" : "<Web Engagement Server host>:<Web Engagement Server secure
port>/server/api/resources/v1/dsl/domain-model.xml",
    "httpEndpoint" : "<Web Engagement Server host>:<Web Engagement Server port>",
    "httpsEndpoint" : "<Web Engagement Server host>:<Web Engagement Server secure port>"
  }]);
 (function () {
    var gt = document.createElement("script");
    gt.setAttribute("async", "true");
    gt.src = ("https:" == document.location.protocol ? "<Web Engagement Server host>:<Web
Engagement Server secure port>" :
    "<Web Engagement Server host>:<Web Engagement Server port>") + "/server/resources/js/
build/GTC.min.js";
    (document.getElementsByTagName("head")[0] || document.body).appendChild(gt);
  })();
</script>
```

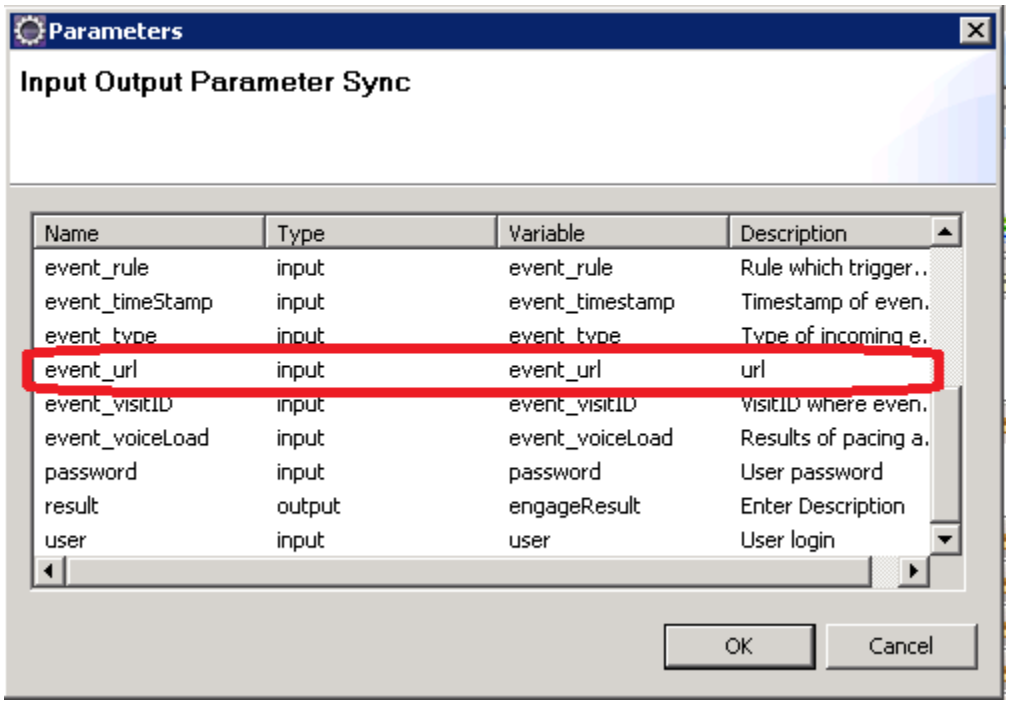
Parsing the address of the page and switching the invitation text

You can get the language code from the page address in the strategy. The page address is passed to the strategy when the rule is triggered and an invitation is generated.

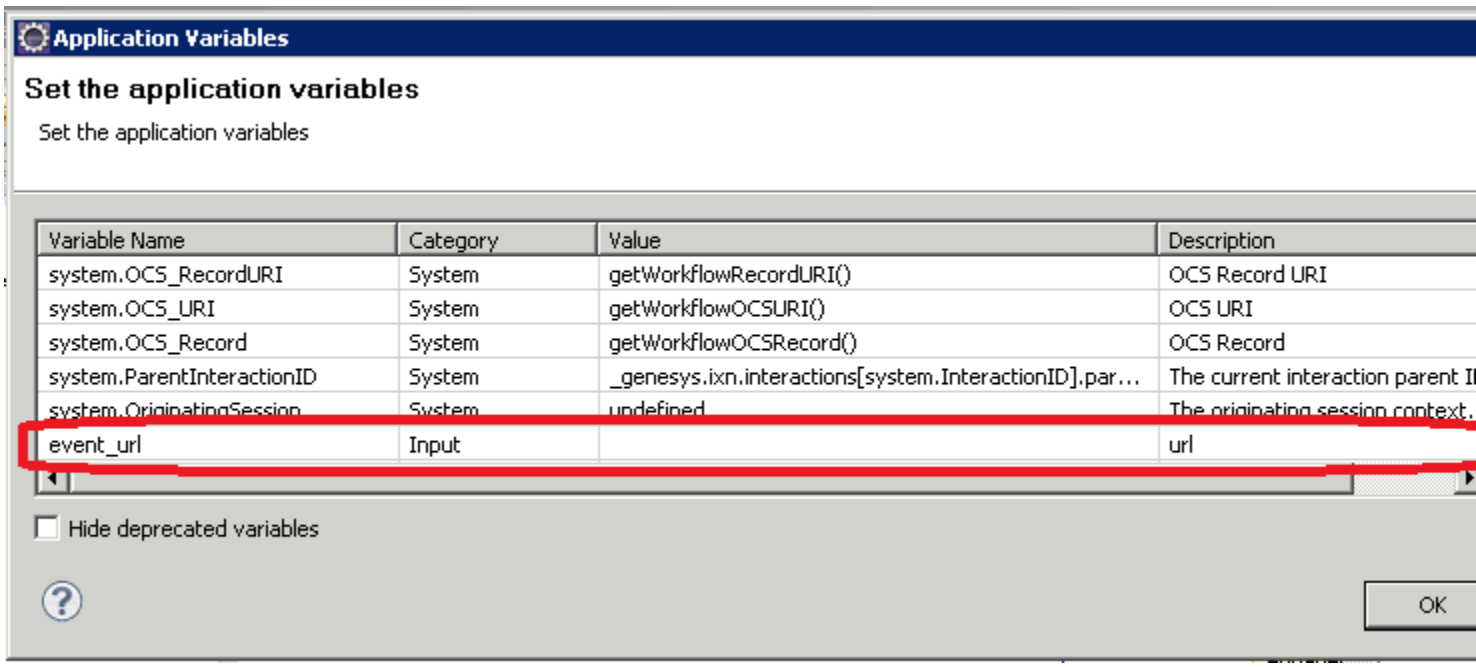
The `event_url` variable is declared and initialized in the `default.workflow` strategy:



The `event_url` variable is then transmitted to the `engage.workflow` strategy:



The following shows the description of the entering variable in the `event_url` in the `engage.workflow` strategy:



The following example shows fetching the language code of the URL address and switching the labels in the FullfillEngagementProfile ECMA Script block in the `engage.workflow` strategy:

```
var language=event_url.substr(-5);
var invitation=" Would you like some help with the selection? Our technical experts are
available to answer questions.";
var acceptBtnText= 'Chat1';
var cancelBtnText = 'No Thanks';
var greetingDefault = 'Hello!';
var greetingMorning =  'Good morning!';
var greetingEvening =  'Good evening!';
var greetingAfternoon = 'Good afternoon!';
var titleChat = 'Chat';

switch (language)
{
case "zh-CN":
  invitation=" ◆◆◆
```

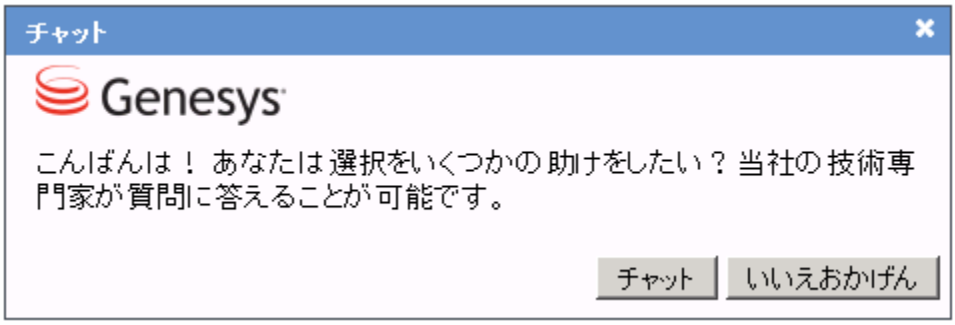
```
    realLocalTime = event_timeStamp - event_timezoneOffset + (new
Date()).getTimezoneOffset()*60000;
    var date = new Date(realLocalTime);
    var hours = date.getHours();
    if (hours < 6)  {
        greeting = greetingDefault;
    } else if (hours < 12) {
        greeting = greetingMorning;
    } else if (hours < 17) {
        greeting = greetingAfternoon;
    } else {
        greeting = greetingEvening;
    }
}

var engageProfile = {
    'visitID': event.visitID,
    'nick_name': profile.FirstName,
    'first_name': profile.FirstName,
    'last_name': profile.LastName,
    'subject': channelName,
    'message':greeting + invitation,
    'time_zone_offset': 8,
    'wait_for_agent' : false,
    'routing_point':sipRoutingPoint,
    'ixn_type': channelType,
    'pageID': event.pageID,
    'inviteTimeout': 30,
    'acceptBtnCaption': acceptBtnCaption,
    'cancelBtnCaption': cancelBtnCaption
};
```

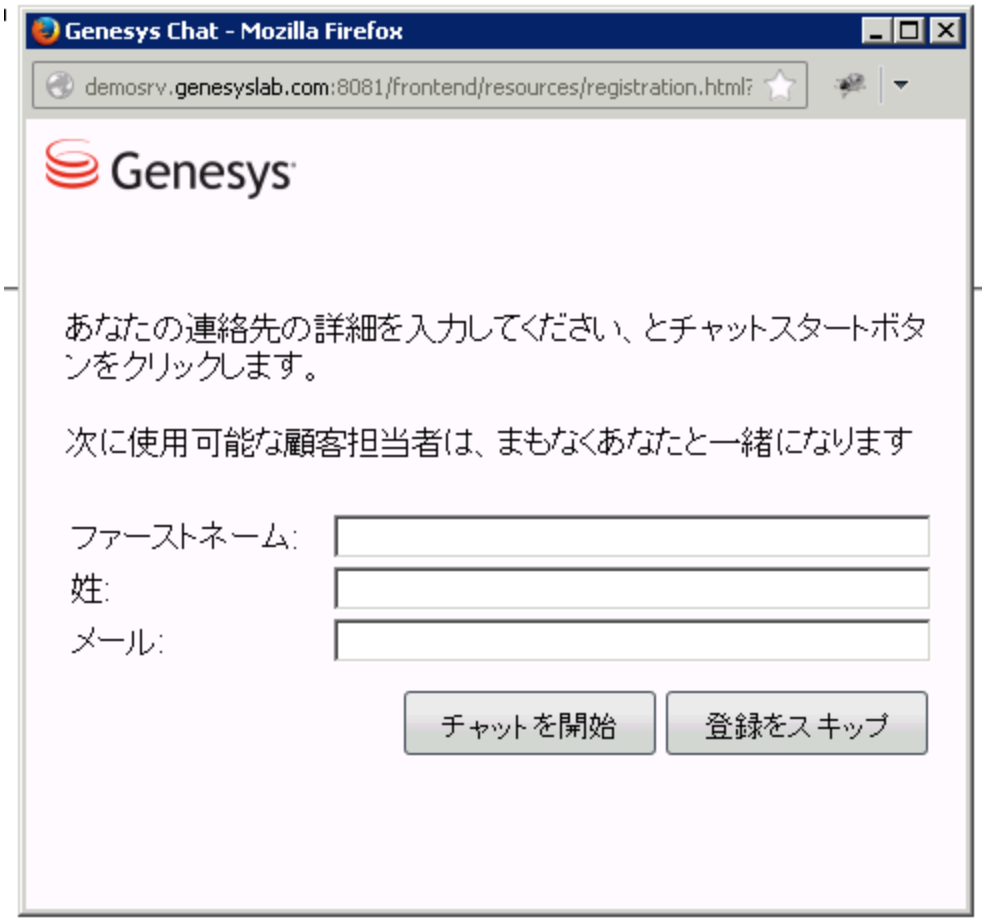
Localized widgets examples

Implementing the code above will result in localized versions of the Web Engagement widgets. For example, if the language is Japanese, the text in the widgets would appear as follows:

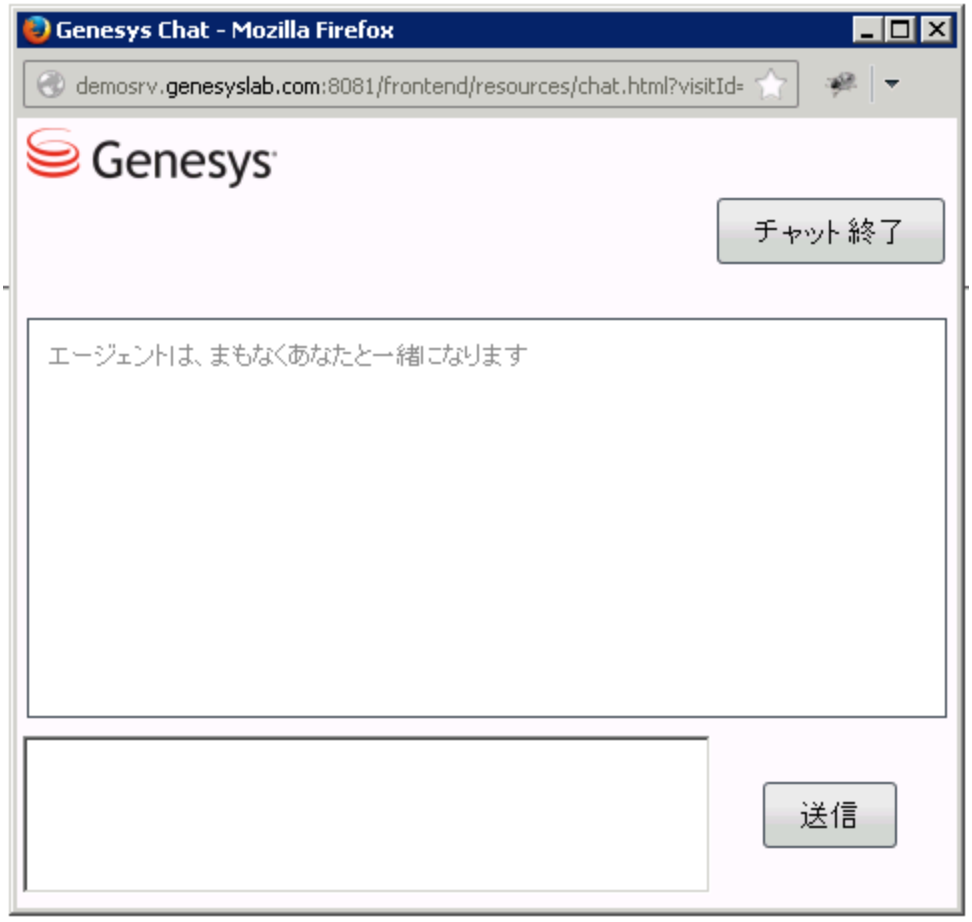
Engagement invitation:



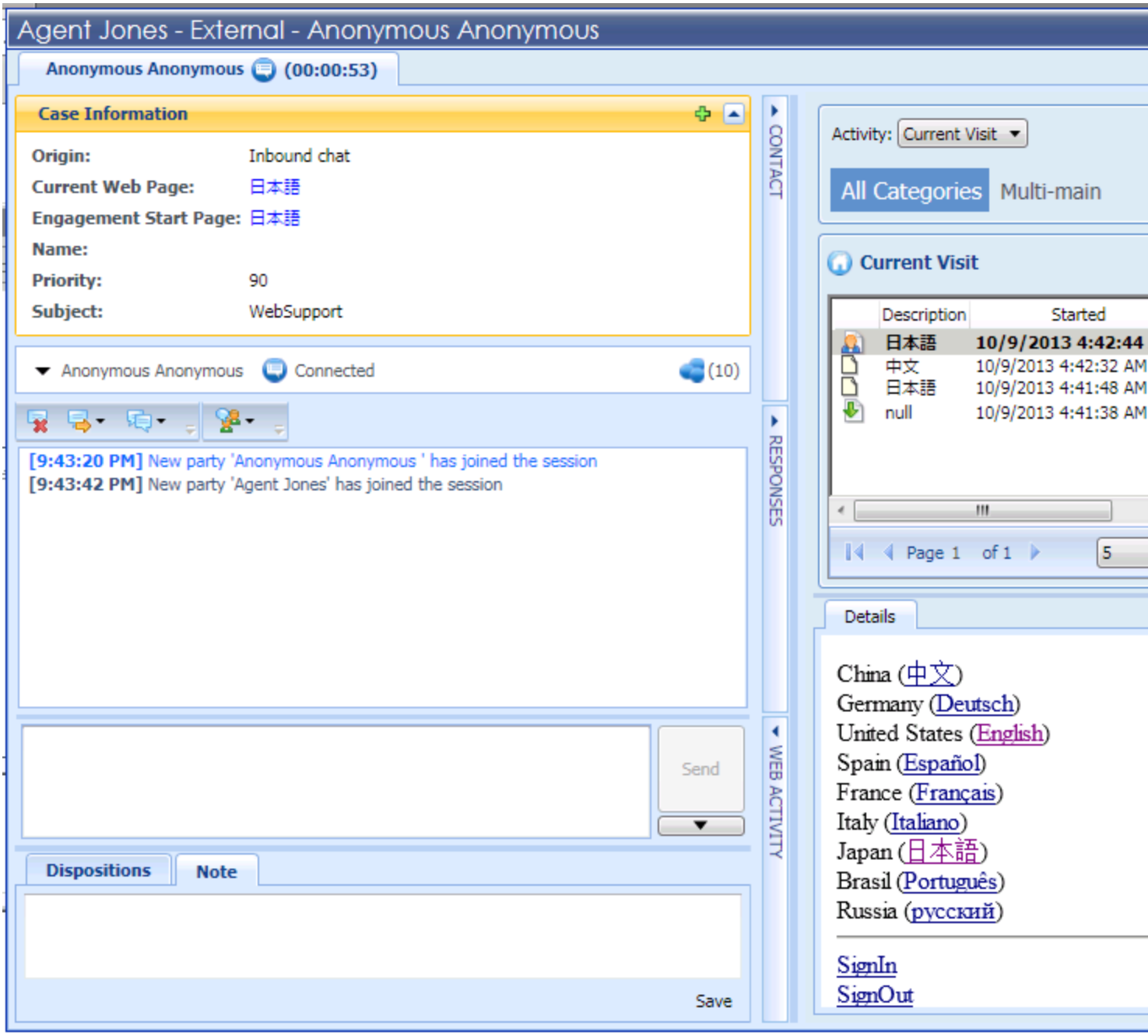
Registration form:


Genesys Chat - Mozilla Firefox
demosrv.genesyslab.com:8081/frontend/resources/registration.html?
Genesys
あなたの連絡先の詳細を入力してください、とチャットスタートボタンをクリックします。
次に使用可能な顧客担当者は、まもなくあなたと一緒になります
ファーストネーム:
姓:
メール:
チャットを開始
登録をスキップ

Chat:



Interaction Workspace:


Agent Jones - External - Anonymous Anonymous
Anonymous Anonymous (00:00:53)
Case Information
Origin: Inbound chat
Current Web Page: 日本語
Engagement Start Page: 日本語
Name:
Priority: 90
Subject: WebSupport
Anonymous Anonymous Connected (10)
[9:43:20 PM] New party 'Anonymous Anonymous ' has joined the session
[9:43:42 PM] New party 'Agent Jones' has joined the session
Send
Dispositions
Note
Save
CONTACT
RESPONSES
WEB ACTIVITY
Activity: Current Visit
All Categories
Multi-main
Current Visit
Description
Started
日本語 10/9/2013 4:42:44
中文 10/9/2013 4:42:32 AM
日本語 10/9/2013 4:41:48 AM
null 10/9/2013 4:41:38 AM
Page 1 of 1
Details
China (中文)
Germany (Deutsch)
United States (English)
Spain (Español)
France (Français)
Italy (Italiano)
Japan (日本語)
Brasil (Português)
Russia (русский)
SignIn
SignOut