Troubleshooting

Genesys Web Engagement 8.5.1

3/14/2022

Table of Contents

Genesys Web Engagement Troubleshooting Guide

Welcome to the *Genesys Web Engagement 8.5.1 Troubleshooting Guide*. This guide provides solutions to common problems in Genesys Web Engagement. See the summary of chapters below.

ADDP

- Configure an ADDP connection to Config Server

Cassandra and Elasticsearch Indexes

- Tune logging with Cassandra running as a Windows service

CometD

- Tuning the CometD timeout interval to optimize chat channels for mobile device activity

InTools

- How do I create a new DSL event when I work with an instrumented website?
- How do I get rid of this "console" panel under the GWE DSL Editor?
- When I refresh and click `Load from page` I see a warning message: "InTools could not find DSL on the current page"
- InTools doesn't communicate with the current page. I don't see events in the Event Console or I see a warning message: "Unable to load DSL from current page"

IPv6

- Why can't my GWE servers connect to other Genesys servers through IPv6?

Rules

- Working with simultaneous events
- Prevent OutOfMemory exceptions when using multiple conditions on the same event

SCXML Strategies

- Hosting your Engagement Logic routing strategy on another server

ADDP

Configure an ADDP connection to Config Server

Web Engagement Server normally uses standardized Genesys ADDP properties to connect to Genesys servers. However, a successful ADDP connection to Config Server may require extra configuration information, especially when a Config Server connection is not explicitly specified for the Web Engagement Cluster application. This is because the ADDP configuration information is stored on the Config Server—so you can't read it until after you have connected.

To configure your Web Engagement Server to use ADDP for its initial Config Server connection, you must modify your **launcher.ini** or **setenv.sh** file:

1. Navigate to the installation directory for your Web Engagement Server and open the **server\launcher.ini** file (for Windows) or the **server\setenv.sh** file (for Linux) with a text editor.
2. For Windows:
    1. Find the line that starts with `-Dwcc.primaryConfServer=`. It configures parameters of connection to the config server, including ADDP.
    2. Change parameters to the level you want
3. For Linux:
    1. Find the line that starts with `USE_ADDP=` and set the value to `true`
    2. Find the line that starts with `ADDP_TRACE_MODE=` and set the value to `local`, `remote`, or `both`. This will enable logging for ADDP-related events.
4. Save your changes.
5. Restart Web Engagement Server to apply changes.

Cassandra and Elasticsearch Indexes

Tune logging with Cassandra running as a Windows service

By default, the Cassandra Windows service sends **stdout** to a file, as you can see if you examine the **StdOutput** and **StdError** settings in the **cassandra.bat** and **cassandra.ps1** files (since the `auto` setting indicates that the output should be sent to a file):

```
--StdOutput auto --StdError auto
```

When you install Cassandra as a Windows service, these settings are loaded in the Windows registry under this key:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Apache Software Foundation\Procrun 2.0\cassandra\
parameters\Log]
```

To disable file-based logging, you can carry out either of the following procedures.

Modifying the Windows Service

Start

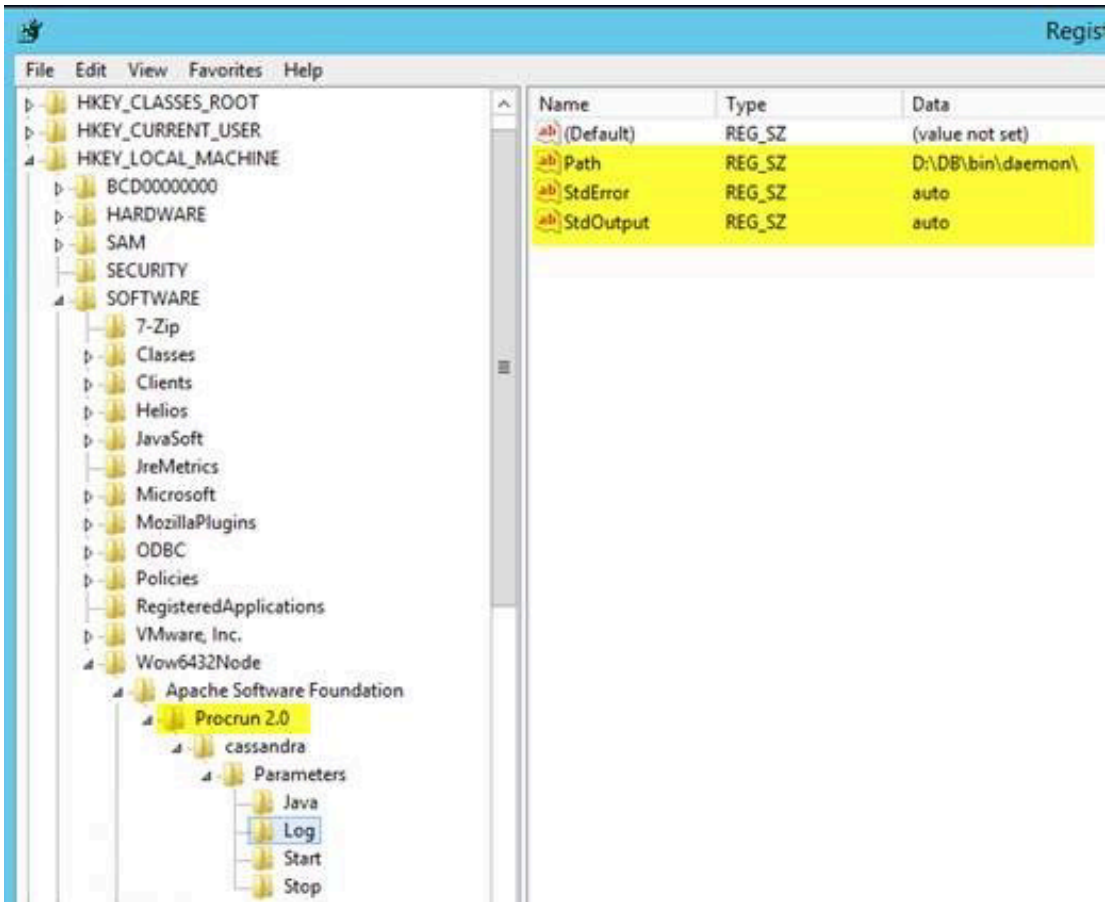
1. Uninstall the Cassandra Windows service.
2. Specify empty values for the **StdOutput** and **StdError** parameters in the **cassandra.bat** or **cassandra.ps1** file.
3. Reinstall the Cassandra Windows service.

End

Editing the Registry

Start

1. Open the Windows Registry Editor
2. Open the appropriate settings under the **[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Apache Software Foundation\Procrun 2.0\cassandra\parameters\Log]** key.



3. Specify empty values for the **StdOutput** and **StdError** parameters.

End

CometD

Tuning the CometD timeout interval to optimize chat channels for mobile device activity and for notifications

When people use their mobile devices, they often need a longer chat timeout interval than they do for their laptops. For example you might start a chat and then:

- Lock your screen for a minute or two
- Switch to another application before you resume your chat

Given that the default value for CometD's maximum timeout interval is 15 seconds, these everyday patterns of mobile device usage can cause your customers to experience unexpected disconnections when you use the Web Engagement Chat Widgets in scenarios involving mobile devices. Although a 15-second timeout interval can be just right for dealing with unstable networks, Genesys recommends that you consider increasing this value to something like 120 or 180 seconds in order to avoid these mobile-related problems.

You can also adjust this interval in situations where your network is experiencing significant delays, since Web Engagement uses CometD as a channel for notification messages.

> **Important**
>
> It is also important to avoid overly large values for this option, which can adversely affect memory consumption and performance on the server side.

Tuning the CometD for multiple iFrames on a web page

The Tracker script receives real-time notifications only on the active web page. This is because, by default, CometD opens only one long poll session per domain connection, per browser. When GWE monitors the active web page along with one or more iFrames, either only the main page or only one iFrame receives notifications from the GWE server. The other instances of Tracker wait for an available long poll connection.

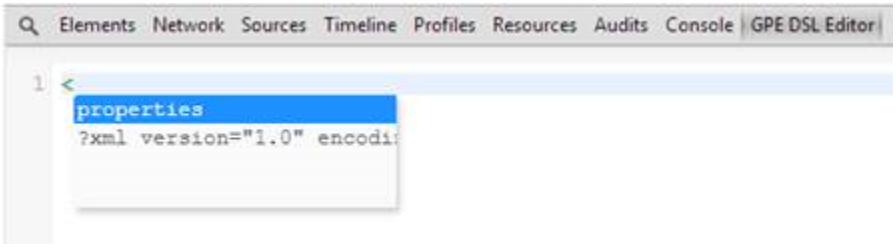
To monitor a web page that contains multiple iFrames, set the maxSessionsPerBrowser option to the number of iFrames on the web page.

> **Important**
>
> Genesys recommends setting the maxSessionsPerBrowser option to the **minimal** required level. For example, if you use Genesys Web Engagement to monitor a web page that contains one iFrame, set the **maxSessionsPerBrowser** option to 2. Do not set it to -1 (unlimited).

InTools

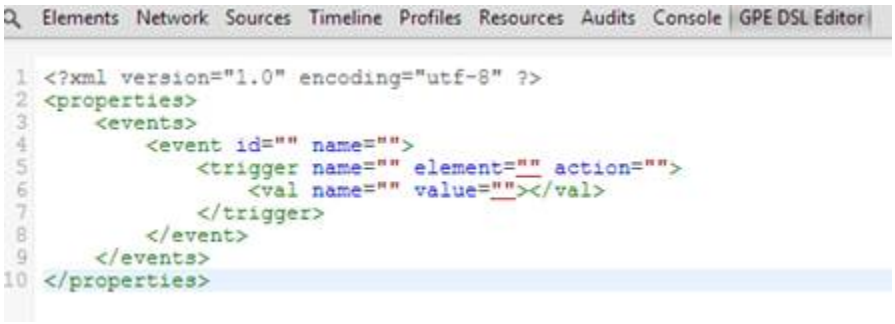
How do I create a new DSL event when I work with an instrumented website?

You can do this by first removing the DSL in the GWE DSL Editor. Type the "<" character and choose `?xml version="1.0" encod...` from the list of code complete options.



The editor display code complete options.

InTools will then create a new DSL:



The new DSL.

You can use the same approach with DSL elements such as event, trigger, value.

How do I get rid of this "console" panel under the GWE DSL Editor?

You can hide this console by using the `Esc` key while InTools is in focus. This shows/hides the "console" panel.

You can also click the `Hide Drawer` button.

Click `Hide Drawer` to show/ hide the "console" panel.

When I refresh and click `Load from page` I see a warning message: "InTools could not find DSL on the current page"

This mean the DSL could not be provided before the Genesys Web Engagement script initialized. When you try to "Load from page", InTools checks that the tracking script is present on the page. If the script is absent, InTools cannot get the DSL from the page.

Try clicking `Load from page` again when the page is fully refreshed.

InTools doesn't communicate with the current page. I don't see events in the Event Console or I see a warning message: "Unable to load DSL from current page"

You might have detached InTools from the current window and closed it. As a result, your InTools does not connect with any page.

Try to close all opened InTools windows and reopen InTools on the current page. You can use F12 on a page that is in focus to reopen the Developer Panel with InTools.

IPv6

Why can't my GWE servers connect to other Genesys servers through IPv6?

If this happens, you should specify an additional JVM option (system property) called **java.net.preferIPv6Addresses** in the **launcher.init** or **setenv.sh** file.

Start

1. Navigate to the installation directory for your Web Engagement Server and open the **server\launcher.ini** file for Windows — or the **server\setenv.sh** file for Linux — with a text editor.
2. For Windows:
   - Find this line: `-Djava.net.preferIPv4Stack=true` and change value of the option to `false`
   - Right after this line add new line: `-Djava.net.preferIPv6Addresses=true`
3. For Linux:
   - Find this line: `ENABLE_IPV6="false"` and set value of the option to `true`
   - Uncomment line: `IP_VERSIONS="4,6"` and specify value of the option to the `"6,4"`
4. Save the file.
5. Restart Web Engagement Server.

End

Rules

Working with simultaneous events

Sequence patterns do not always work in cases where multiple events for the same visit are sent simultaneously.

A sequence pattern is described by the following construction:

```
$event2: Event(eval($event2.getName().equals('...')) && this after $event1)
```

The Web Engagement rules engine can only process patterns like this one if **event2** arrives at least 1 millisecond later than **event1**. However, simultaneously generated events may arrive in the same millisecond, which means that the pattern will not work.

Under a rare set of circumstances, multiple events might be sent simultaneously for the same visit. If this happens in your environment, you need to modify the default GRDT templates, changing your sequence pattern to the following form:

```
$event2: Event(eval($event2.getName().equals('...')) && this after[0ms] $event1)
```

Prevent OutOfMemory exceptions when using multiple conditions on the same event

GRAT only allows you to use 5 parameters in a single conditional statement when creating a linear rule. You may want to work around this limitation and apply more conditional statements to the same event, so that the total number of applied parameters will be greater than 5. In this case, you can use additional **eval** statements, as explained below.

The default Web Engagement GRDT templates contain conditional statements that look something like this:

```
$event2: Event(eval($event2.getName().equals('MyEvent')) && this after $event1)
```

This statement uses the **Event** keyword when evaluating the nested expressions. However, if you use more than 1 conditional statement that uses the **Event** keyword, the Web Engagement rules engine might produce a large number of fake "activation" objects that could lead to OutOfMemory exceptions.

Here is an example of a set of statements that can cause this issue, because each of them uses this keyword:

```
$event1: Event(eval($event1.getName().equals('MyEvent')))
Event(eval($event1.getDataPropertyValue('XXX') != null ))
...
Event(eval($event1.getDataPropertyValue('XXX') != null ))
```

Fortunately, you can use **eval** conditions without needing the **Event** keyword, as shown here:

```
$event1: Event(eval($event1.getName().equals('MyEvent')))
eval($event1.getDataPropertyValue('XXX') != null )
...
eval($event1.getDataPropertyValue('XXX') != null )
```

Since only the first statement uses the **Event** keyword, the rules engine will not create fake "activation" objects, meaning that you can avoid this type of OutOfMemory problem.

SCXML Strategies

Hosting your Engagement Logic routing strategy on another server

By default, your Engagement Logic routing strategy is hosted on your Web Engagement Server. If you want to host it on another web server, follow these steps:

Start

1. Locate the routing strategy that you want to host on the new server. By default, the Engagement Logic routing strategy is created and modified in the folder ***Web Engagement Root*\apps\<*your application*>\resources_composer_projects\WebEngagement_EngagementLogic**. If you are planning to use this strategy outside of Web Engagement Server, you'll need to copy the **src-gen** and **include** sub-folders and their contents. You can also use your own strategy.
2. Move the routing strategy to the desired location.
3. Modify the Enhanced Routing objects whose names start with **Webengagement_**. These objects are usually located in the **Scripts** folder.
4. Modify the **url** option from the **Application** section of the Annex so that it points to the strategy's new location.

End

Speed up processing of webengagement Open Media interactions

Open Media interactions of the webengagement type may be collected in the Interaction Queue. At times, they can be processed more slowly than expected. As a result, visitors may leave pages where hot leads are triggered before notifications are sent from Engagement Logic strategy to the browser (through the GWE). For example, this may happen when GWE produces a very large number of hot leads in a short amount of time.

> **Tip**
> Genesys recommends that you avoid triggering an excessive number of hot leads. For example avoid producing a hot lead on every page browsed by a visitor.

Symptom

Too many webengagement Open Media interactions are collected in the Interaction Server database, and they are then pulled too slowly.

Possible solutions

1. Avoid generating unnecessary hot leads. (For example, do not engage customers on each and every page.)
2. Optimize the `mcr-pull-interval` option of the Orchestration Server. For more information about the Orchestration Options, see Application-Level Options.
   - The default setting of 1000 causes ORS to pull one interaction per second.
   - If you specify a value of 100, you increase pulling by 10 times. Please consult Orchestration Server specialists before you modify the production environment.
3. Rework your Engagement Logic SCXML strategy so that it uses enhanced pulling. For more information, see Composer Help.