This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Genesys Pulse Deployment Guide

Genesys Pulse User Interface Extensions

4/20/2026

Contents

Genesys Pulse User Interface Extensions

Introduction

This article describes Genesys Pulse user interface (UI) extensions. The main purpose of these extensions is to support the 3rd-party widgets which are developed separately and installed as a plug-in. In addition, you can load external .js and .css files in order to customize existing widgets.

Important

- The Genesys Pulse extension API is supported on a best-efforts basis. Any customer attempting to use or develop extensions must have expertise to do so and agree to troubleshoot them on their own. This API is subject to change in future versions at Genesys' discretion without prior notice.
- Be extra cautious with extensions that are loading third-party JavaScript libraries as they can break Genesys Pulse functionality.

How to Create Your Own Plug-in

Starting with release 8.5.108, you can create the Genesys Pulse plug-in skeleton using the `plugin-generator.js` utility script from the **plugin-sdk\libs** directory of the Genesys Pulse installation package. This script is tested with the latest nodejs version 8, but should also work with the nodejs version 6.5 or higher.

For example:

```
node plugin-generator.js
Enter plugin name (full plugin name will be pulse-plugin-<name>):
> My Funnel
Enter plugin version:
> 1.0.0
Enter plugin description:
> My Description
Creating ./pulse-plugin-my-funnel/pom.xml
Creating ./pulse-plugin-my-funnel/src/main/resources/META-INF/applicationContext.xml
Creating ./pulse-plugin-my-funnel/src/main/java/com/genesyslab/wbrt/plugin/
PulsePluginMyFunnel.java
Creating ./pulse-plugin-my-funnel/src/main/resources/web/manifest.js
Creating ./pulse-plugin-my-funnel/src/main/resources/web/pulse.manifest.js
Plugin structure has been created
```

After that, you need to insert your extension implementation into a generated pulse.manifest.js file.

To build a plug-in you need to have Java and Maven installed. Maven dependencies are located in the **plugin-sdk\libs** directory of the Genesys Pulse installation package. You can install them by running the installation script from the same directory.

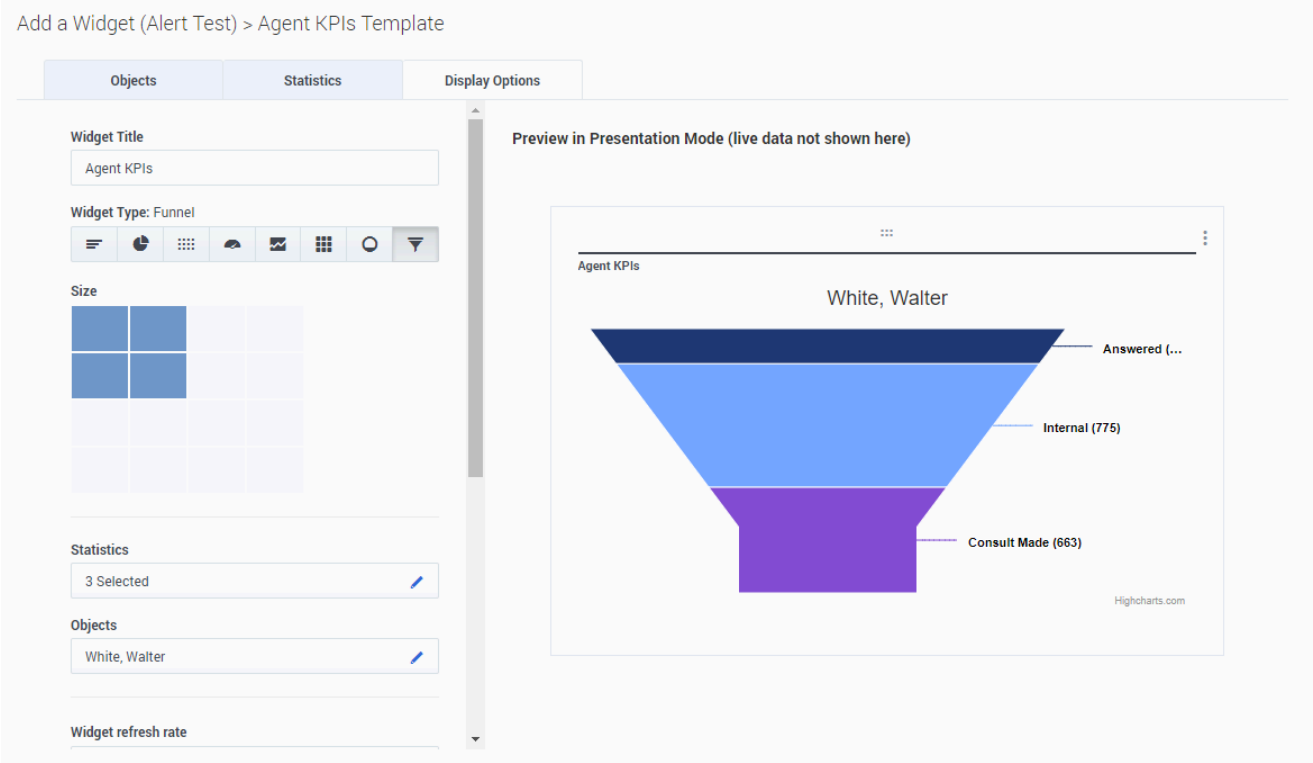
After that you are ready to build a plug-in:

```
cd pulse-plugin-my-funnel
mvn clean install
```

Now the plug-in is ready to be deployed into Genesys Pulse. Copy it to the **<PULSE_PATH>/webapp/WEB-INF/lib/** and restart Genesys Pulse.

> ### Tip
>
> To speed up the development process you do not need to rebuild and deploy your plug-in after each change. You can modify an already deployed `pulse.manifest.js` file in the **<PULSE_PATH>/webapp/plugins/<your_plugin_name>/** directory. To apply any change you just need to reload the Genesys Pulse page. When you are finished, ensure that you copy the final `pulse.manifest.js` file back to your project directory located at **<your_plugin_path>/src/main/resources/web/**, otherwise you might lose all your changes when Genesys Pulse is restarted.

Now the new Funnel chart is available among other widget types:



Genesys Pulse User Interface Extension API

Each `pulse.manifest.js` should contain one or more immediately-invoked function expression (IIFE) and use the global `window.pulse` function to register IIFEs.

The below example shows the registering of a custom widget which uses the `Highcharts` library to draw a custom chart.

```
(function() {
    pulse.extension({
        type: 'WIDGET',             // type of extension allows us to add more extension
types in the feature
        apiVersion: '9.0.0',        // version of api used in the extension
        id: 'CustomWidgetOne',      // unique extension id, should not clash with other
deployed extensions
        label: 'Custom Widget One', // label displayed in Display Options of the Widget Wizard
        icon: 'icon-app-chart',     // icon displayed in Display Options of the Widget Wizard
        require: [                  // javascript or css files to be loaded for the extension
work
                                    // use object when your library exposes global variable
                                    // no need to load d3 (version 3.5.17), jQuery,
underscore - they are already loaded by Genesys Pulse
                                    // to avoid loading library from CDN put library side by
side with pulse.manifest.js and provide URL like "../pulse-plugin-name/library.js"
             {Highcharts: "https://code.highcharts.com/highcharts.js"},
             "https://code.highcharts.com/highcharts-more.js"
        ],
        render: function (element, data, options) { // key function for rendering widget
content
            ... //put your rendering code here
            return true; // return true when widget content is rendered or false when widget
cannot be rendered
        },
        resize: function (element, data, options) { // function to be called when widget
being resized
            ... //put your rendering code here
        },
        constraints : { // constraints for widget configuration
           dashboardSupport: true, // allows to select widget on dashboard, enabled by default
           wallboardSupport: true, // allows to select widget on wallboard, disabled by
default
           size: { // define min and max size for the widget
              minX: 1, // min horizontal size
              minY: 1, // min vertical size
              maxX: 1, // max horizontal size
              maxY: 2  // max vertical size
           },
           objects: { // define the min and max objects which can be selected by the users to
adjust the real-state of the visualization
              min: 1, // require at least one object selected
              max: 1  // allow to select no more than one object
           },
           statistics: { // define the min and max stats which can be selected by the users
to adjust the real-state of the visualization
              min: 1, // require at least one statistic selected
              max: 10 // allow to select no more than ten statistic
           }
        },
        containerClass: "my-chart", // optional, specify custom css class for widget container
        containerStyle: {           // optional, overwrite widget container style
           "padding-bottom": "10px"
        }
```

```
    });
}();
```

The important **render** function accepts three arguments:

- **element**—HTMLElement, where the content of a widget should be rendered.
- **data**—the data from the snapshot formatted for easier use by new developers.
- **options**—additional options, such as current selected statistics and objects, widget size, current locale.

The **render** function is called each time the content of a widget is redrawn with new **data**. When a widget is resized the **resize** function is called with the same three arguments.

The **data** object has the following format:

```
{
  statistics: [{         // array with all statistics selected in Statistics of the Widget
Wizard
    format: "time",      // statistic format
    id: "Ready_Time",    // statistic internal name
    label: "Ready Time", // display name of statistic
    ranges: {            // describes threshold ranges for particular statistic
      green: {
        from: 100,
        color: "green"
      },
      orange: {
        to: 100,
        from: 20,
        color: "yellow"
      },
      red: {
        to: 20,
        color: "red"
      }
    },
    values: [ // array with statistic values per each object, values are ordered according to
objects order in objects array
      6470,
      6435,
      6467
    ]
  }],
  objects: [{ // array with all objects selected in Objects of the Widget Wizard
      id: 103,
      label: "Master, Yoda"
    },
    {
      id: 104,
      label: "Darth, Vader"
    },
    {
      id: 9638,
      label: "Luke, Skywalker"
    }
  ],
  history: { // object provides access to historical values of statistic, available since
Genesys Pulse version 9.0.0
    get: function(statisticId, objectId, options) { ... } // method returning array with
historical data for particular statistic and object
    // additional parameters can be passed via options object:
```

```
    // options.start - optional, allows to specify left time boundary of historical data
represented either by Date object or by milliseconds
    // example of method call result: [{time: 1517845138005, value: 1}, {time: 1517845156610,
value: 42}, ...]
  }
}
```

This format is self-describing, easy to understand and easy to transform into input for popular charting libraries like `Highcharts`.

The **options** object has the following format:

```
{
  selectedObjects: [ 103, 104 ],        // array with ids of objects selected in Display
Options to show in chart
  selectedStatistics: [ "Ready_Time" ], // array with ids of statistics selected in Display
Options to show in chart
  theme: {                              // object describes current selected color theme,
could be used to pick Genesys approved colors for charts
    backgroundColor:"#fdfdfd",
    chartColors: [ ... ]
    rangeColors: {
      green:"#4ac764"
      orange:"#f8a740"
      red:"#ea4f6b"
   }
}
```

How to Change Basic Styles

Starting with release 8.5.106, you can customize .js and .css files to apply a new set of colors for the Genesys Pulse instance.

You can place js/css files into the Genesys Pulse folder or **styles/scripts** subfolders. The options for loading custom js/css files are as follows:

- `pulse/load_css_custom=/pulse/styles/custom.css`
- `pulse/load_js_custom=/pulse/scripts/custom.js`

You can specify many options with `load_css/load_js` prefixes:

- `pulse/load_css_<style_name_1>=/pulse/styles/custom_1.css`
- `pulse/load_css_<style_name_n>=/pulse/scripts/custom_n.css`
- `pulse/load_js_<script_name_1>=/pulse/styles/custom_1.js`
- `pulse/load_js_<script_name_n>=/pulse/scripts/custom_n.js`

The options do not require Genesys Pulse restart.

> **Tip**
>
> Files placed in Genesys Pulse folders or **styles/scripts** subfolders can be overwritten during the upgrade. It is better to add these files to a different directory accessible by the http(s) protocol.

You can use a full URL when files are hosted on another web server:

- pulse/load_css_custom=http://<host>:<port>/mystyle.css
- pulse/load_js_custom=http://<host>:<port>/myscript.js

There are samples in the **examples** folder where Genesys Pulse is installed, in the **custom-css-example** and **custom-js-example** subfolders. These samples are intended for advanced users, who can do their own customization of provided examples.

The font size in the `Text` widgets is designed to scale the font according to the content. However, you can use a custom .css to customize it if you have a competent web designer or with the help of Genesys Professional Services.