Widget BUS Guide

Genesys Widgets 8.5

12/30/2021

Table of Contents

Genesys Widgets Bus Guide

The Widget Bus API Guide provides reference information that you can use to control the Genesys WebChat Widget, including:

- Overview of the Widget Bus API
- Extensions

Widget Bus API Overview

The Widget Bus (CXBus) is a publish/subscribe/command bus designed for User Interfaces. It allows different UI components and controllers to communicate with each other and bind business logic together into a larger, cohesive product.

CXBus supports publishing and subscribing arbitrary events with data over the bus and any other plugin on the bus can subscribe the that event. Publications and subscriptions are loosely bound so that you can publish and subscribe to any event without that event explicitly being available. This allows for plugins to lazy load into the bus or provide conditional logic in your plugins that wait for other plugins to be available.

The full CXBus API reference for each Widget/Plugin is available here: Widgets Reference

CXBus events and commands are executed asynchronously using deferred methods and promises. This allows for:

- Better performance
- Standardized Pass/Fail handling for all commands
- Command promises are not resolved until the command is finished, including any nested asynchronous commands that command may invoke. This gives you assurance that the command completed successfully and the timing of your follow-up action will occur at the right time
- Permissions: CXBus provides metadata in every command call including which plugin called the command and at what time. This allows for plugins to selectively allow/deny invocation of commands.

The Bus is accessible via three methods:

- **Global Access** (Available at runtime after Genesys Widgets have initialized. See **onReady** examples in the Use Cases below).

```
window._genesys.widgets.bus
```

This method is very useful for manually triggering commands on the bus using the JavaScript console in your browser. It is also accessible by your own custom programs at runtime for easy integration with Genesys Widgets.

- **Genesys Widgets onReady callback**

```
window._genesys.widgets.onReady = function(CXBus){

    // Use the CXBus object provided here to interface with the bus
    // CXBus here is analogous to window._genesys.widgets.bus
};
```

- **Extensions**

You can define your own plugins/widgets that interface with Genesys Widgets. For more information, please see Extensions.

Use Cases

Important

For the following examples let us assume we are working within the **onReady** callback function.

Example 1: Subscribing to an Event

Format: `CXBus.subscribe("event name and path", function(e){})`

Example:

```
CXBus.subscribe("WebChat.ready", function(e){

    // interact with the WebChat widget now that it is initialized (ready)
});
```

Example 2: Invoking a Command

Format: widgetBus.command("command name and path", options).done(function(e){}).fail(function(e){})

Example:

```
CXBus.command("WebChat.open").done(function(e){

    // success scenario
    // the value of e depends on the command being called
    // Review the API reference for the particular command you are calling

}).fail(function(e){

    // failure scenario: error, exception, improper arguments
    // the value of e is typically an error string or an AJAX response object
    // Review the API reference for the particular command you are calling
})
```

Genesys Widgets Extensions

Genesys Widgets allows 3rd parties to create their own plugins/widgets to extend the default package. Extensions are an easy way to define your own while utilizing the same resources as core Genesys Widgets.

Defining Extensions

Extensions are defined at runtime before Genesys Widgets load. You can define them inline or include extensions in separate files, either grouped or separated.

Important

Define/Include your extensions AFTER your Genesys Widgets configuration object but BEFORE you include the Genesys Widgets JavaScript package

```
<script>

if(!window._genesys.widgets.extensions){

    window._genesys.widgets.extensions = {};
}

window._genesys.widgets.extensions["TestExtension"] = function($, CXBus, Common){

    var oTestExtension = CXBus.registerPlugin("TestExtension");

    oTestExtension.subscribe("WebChat.opened", function(e){});

    oTestExtension.publish("ready");

    oTestExtension.command("WebChat.open").done(function(e){

          // Handle success return state

    }).fail(function(e){

          // Handle failure return state
    });

    oTestExtension.registerCommand("demo", function(e){

          // Command execution here
    });
};

</script>
```

Make sure that the "extensions" object exists and always include this at the top of your extension definition.

```
if(!window._genesys.widgets.extensions){

window._genesys.widgets.extensions = {};
}
```

Create a new named property inside the "extensions" object and define it as a function. When Genesys Widgets initializes it will step through each extension and invoke each function, initializing them. Genesys Widgets will share resources as arguments. These include: jQuery, CXBus, and Common (common UI utilities).

```
window._genesys.widgets.extensions["TestExtension"] = function($, CXBus, Common){
```

Creating a new CXBus plugin

Inside the extension function is where you create a new CXBus plugin. You can use this CXBus plugin to interface with other Genesys Widgets. You can add your own UI controller logic in here or simply use the extension to connect an existing UI controller to the bus (for example, share its API over the bus and coordinate actions with events).

Registering a new plugin on the bus creates a new, unique namespace for all your events and commands. In this example, the namespace "cx.plugin.TestExtension" is created:

```
var oTestExtension = CXBus.registerPlugin("TestExtension");
```

Important

When referring to other namespaces, like "cx.plugin.TestExtension", it is not necessary to include the "cx.plugin." prefix. It is optional and implied. You can subscribe to events or call commands using the full or truncated namespce.

Use Cases

Extensions are like any other Genesys Widget. You can publish, subscribe, call commands, or register your own commands on the bus. You can interface with other widgets on the bus for more complex interactions. The following examples demonstrate how you can make extensions work for you.

Example: subscribing to an event.

```
oTestExtension.subscribe("WebChat.opened", function(e){});
```

Example: publishing an event.

Publishes the event "TestExtension.ready" on the bus.

```
oTestExtension.publish("ready", {arbitrary data to include});
```

Example: calling a command.

Commands are deferred functions. You must handle their return states asynchronously.

```
oTestExtension.command("WebChat.open", {any options required}).done(function(e){

    // Handle success return state
    // "e", the event object, is a standard CXBus format
    // Any return data will be available under e.data

}).fail(function(e){

    // Handle failure return state
    // "e", the event object, may contain an error message, warning, or AJAX response object
});
```

Example: Registering a command.

Creates a new command under your namespace that you or other widgets can call.

"e", the event object, is a standard CXBus format

- e.data = options passed into command when being called.
- e.commander = the namespace of the widget that called this command.
- e.command = the name of the command being called.
- e.time = timestamp when the command was called.
- e.deferred = the deferred promise created for this command call. You MUST always resolve or reject this promise using e.deferred.resolve() or e.deferred.reject(). You may pass any arbitrary data into either resolution state.

```
oTestExtension.registerCommand("demo", function(e){

    // Command execution here
});
```