



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.

Widgets API Reference

Table of Contents

Widgets bus (CXBUS)	
Widgets Bus API overview	6
Widgets-Core	
App	21
Common	40
Overlay	64
Toaster	70
WindowManager	75
Service plugins	
CallbackService	80
StatsService	89
WebChatService	98
UI plugins	
Calendar	129
Callback	140
CallUs	163
ChannelSelector	172
Console	186
SideBar	192
WebChat	204
Bridge plugins	
Engage	246
Extensions	
Genesys Widgets extensions	262
Videos	
Genesys Widgets videos	266

Contents

- [1 Widgets bus \(CXBus\)](#)
- [2 Widgets-Core](#)
- [3 Service plugins](#)
- [4 UI plugins](#)
- [5 Bridge plugins](#)
- [6 Extensions](#)
- [7 Videos](#)

The Widgets API Reference covers all of the commands and events for each widget, and covers how to configure and localize each one.

Related documentation:

-

The APIs are divided into the following categories, as discussed in the article on How Widgets Works. There is also an article that explains how to get started with Genesys Widgets.

Widgets bus (CXBUS)

- [Widget Bus API Overview](#)

Widgets-Core

- [App](#)
- [Common](#)
- [Overlay](#)
- [Toaster](#)
- [WindowManager](#)

Service plugins

- [CallbackService](#)
- [StatsService](#)
- [WebChatService](#)

UI plugins

- [Calendar](#)
- [Callback](#)
- [CallUs](#)
- [ChannelSelector](#)
- [Console](#)
- [SideBar](#)
- [WebChat](#)

Bridge plugins

- Engage

Extensions

- Genesys Widgets Extensions

Videos

- Genesys Widgets supplemental videos

Widgets Bus API overview

Contents

- **1 Overview**
 - 1.1 Global access
 - 1.2 Genesys Widgets onReady callback
 - 1.3 Extensions
- **2 CXBus Reference**
 - 2.1 CXBus.command
 - 2.2 CXBus.configure
 - 2.3 CXBus.loadFile
 - 2.4 CXBus.loadPlugin
 - 2.5 CXBus.registerPlugin
- **3 CXBus Plugin Interface Reference**
 - 3.1 oMyNewPlugin.registerCommand
 - 3.2 oMyNewPlugin.registerEvents
 - 3.3 oMyNewPlugin.subscribe
 - 3.4 oMyNewPlugin.publish
 - 3.5 oMyNewPlugin.republish
 - 3.6 oMyNewPlugin.publishDirect
 - 3.7 oMyNewPlugin.command
 - 3.8 oMyNewPlugin.before
 - 3.9 oMyNewPlugin.registry
 - 3.10 oMyNewPlugin.subscribers
 - 3.11 oMyNewPlugin.namespace
 - 3.12 oMyNewPlugin.ready

- Developer

Learn about the bus that all widgets components are built on.

Related documentation:

-

Overview

Genesys Widgets is built on top of the CXBus messaging bus. CXBus uses the publish-subscribe model to facilitate communication between the Widgets components, all of which are *plugins* that can both *publish* events on the bus and *subscribe* to the events they are interested in.

With the help of the Widgets-Core plugins, CXBus makes it possible to combine the logic implemented by user interface plugins, service plugins, and utility plugins into cohesive products that can provide chat sessions, schedule callbacks, and so on.

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 so they can wait for other plugins to be available.

CXBus events and commands are executed asynchronously using deferred methods and promises. This allows for better performance and 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. As for 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.

You can use three methods to access the Bus:

- Global access
- Genesys Widgets onReady callback
- Extensions

Global access

QuickBus

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

For quick access to call commands on the bus, you can access the **QuickBus** instance after Genesys Widgets loads. QuickBus is a CXBus plugin that is exposed globally for your convenience. Typical use cases for using QuickBus are for debugging or calling a command when a link or button is clicked. Instead of creating your own plugin, you can use QuickBus to add the click handler inline in your

HTML.
Example:

[Open WebChat](#)

Global CXBus

CXBus is available as a global instance named "CXBus" (or `window.CXBus`). Unlike QuickBus, this is not a plugin but CXBus itself.

CXBus has been updated to include a "command" method that allows you to execute a command directly from the CXBus instance.

Example:

```
CXBus.command("WebChat.open");
```

You can use this, like QuickBus, for debugging or setting up click events.

Genesys Widgets onReady callback

Genesys Widgets provides an "onReady" callback function that you can define in your configuration. This will be triggered after Genesys Widgets initializes. QuickBus is provided as an argument in this function, but you may also access CXBus globally in your function.

```
window._genesys.widgets.onReady = function(QuickBus){  
    // Use the QuickBus plugin provided here to interface with the bus  
    // QuickBus 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 Genesys Widgets extensions.

CXBus Reference

The CXBus instance is exposed globally (`window.CXBus`) and has several methods available:

- `CXBus.command`
- `CXBus.configure`
- `CXBus.loadFile`
- `CXBus.loadPlugin`
- `CXBus.registerPlugin`

CXBus.command

Calls a command on the bus under the namespace "CXBus". Use this to quickly and easily call

commands without needing to generate a unique plugin interface object first.

Example

```
CXBus.command("WebChat.open", {});
```

Arguments

Name	Type	Description
Command name	string	The name of the command you wish to execute.
Command options	object	Optional: You may pass an object containing properties that the command will accept. Refer to the documentation on each command to see what options are available.

Returns

Always returns a promise. You can define `done()`, `fail()`, or `always()` callbacks for every command.

CXBus.configure

Allows you to change configuration options for CXBus.

Example

```
CXBus.configure({debug: true, pluginsPath: "/js/widgets/plugins/"});
```

Arguments

Name	Type	Description
Configuration options	object	An object containing properties, similar to command options. In this object you can change configuration options for CXBus.

Configuration options

Name	Type	Description
debug	boolean	Enable or disable CXBus logging in the javascript console. Set to true to enable; set to false to disable. Default value is false .
pluginsPath	string	The location of the Genesys Widgets "plugins" folder. Example: <code>"/js/widgets/plugins/"</code> The default value here is <code>""</code> . This configuration option is used for lazy loading plugin files.

Name	Type	Description
		Be sure to configure this option when using Genesys Widgets in lazy loading mode.
pluginMap	object	<p>Used to change the target JS file for each plugin or to add a new plugin.</p> <p>Example:</p> <pre>{sendmessage: "https://www.yoursite.com/ plugins/custom- sendmessage.js"}</pre> <p>CXBus will automatically lazy load plugins defined in this object when something tries to call a command on that plugin.</p> <p>For instance, if SendMessage.open is called and SendMessage isn't loaded, CXBus will fetch it from the default "plugins/" folder. If you want to load a different SendMessage widget, you can override the default URL of the JS file associated with "sendmessage".</p> <p>You can also prevent a plugin from loading by mapping it to false.</p> <p>Example:</p> <pre>{sendmessage: false}</pre> <div><p>Important</p><ul style="list-style-type: none">Any number of plugins can be included in this object.Only works when using the lazy-loading method of initializing Widgets.Not intended to be used to load different versions of Genesys Widgets plugins.Intended to be used along with the proper pluginsPath configuration. Do not use pluginMap method separately.</div>

Returns

This method returns nothing.

CXBus.loadFile

Loads any javascript file.

Example

```
CXBus.loadFile("/js/widgets/plugins/webchat.min.js");
```

Arguments

Name	Type	Description
File path	string	Loads a javascript file based on the file path specified.

Returns

Always returns a promise. You can define `done()`, `fail()`, or `always()` callbacks. When the file loads successfully, `done()` will be triggered. When the file fails to load, `fail()` will be triggered.

CXBus.loadPlugin

Loads a plugin file from the configured "plugins" folder.

Example

```
CXBus.loadPlugin("webchat");
```

Arguments

Name	Type	Description
Plugin name	string	Loads a plugin from the "plugins" folder by name (configured by the "pluginsPath" option). Plugin names match their CXBus namespaces but are lowercase. Example: To load WebChat, use "webchat". You can refer to the files inside the "plugins" folder as well. The first part of the file name will be the name you use with this function. Example: Use "webchat" to load "webchat.min.js".

Returns

Always returns a promise. You can define `done()`, `fail()`, or `always()` callbacks. When the plugin loads successfully, `done()` will be triggered. When the plugin fails to load, `fail()` will be triggered.

CXBus.registerPlugin

Registers a new plugin namespace on the bus and returns a plugin interface object. You will use the plugin interface object to publish, subscribe, call commands, and perform other CXBus functions.

Example

```
var oMyNewPlugin = CXBus.registerPlugin("MyNewPlugin");
```

Arguments

Name	Type	Description
CXBus plugin namespace	string	The namespace you want to reserve for your plugin.

Returns

If the namespace is not already taken, it will return a CXBus plugin interface object configured with the selected namespace. If the namespace is already taken, it will return false.

CXBus Plugin Interface Reference

When you register a plugin using CXBus.registerPlugin(), it returns a CXBus Plugin Interface Object. This object contains many methods that allow you to interact with other plugins on the bus.

Let's start with the assumption that we've created the below plugin interface:

```
var oMyNewPlugin = CXBus.registerPlugin("MyNewPlugin");
```

oMyNewPlugin.registerCommand

Allows you to register a new command on the bus for other plugins to use.

Example

```
oMyNewPlugin.registerCommand("test", function(e){
    console.log("'MyNewPlugin.test' command was called", e)
    e.deferred.resolve();
});
```

Arguments

Name	Type	Description
Command name	string	The name you want for this command. When other plugins call your command, they must specify the namespace as well.

Name	Type	Description
		Example: "test" is called on the bus as "MyNewPlugin.test".
Command function	function	The command function that is executed when the command is called. This function is provided an Event Object that contains metadata and any options passed in.

Event object

Name	Type	Description
time	number (integer time)	The time the command was called.
commander	string	The name of the plugin that called your command. Example: If your plugin called a command, the value would be "MyNewPlugin". You can use this information to create plugin-specific logic in your command.
command	string	The name of this command. Example: "MyNewPlugin.test". This can be useful if you are using the same function for multiple commands and need to identify which command was called.
deferred	deferred promise object	When a command is called, a promise is generated. You must resolve this promise in your command without exception. Either execute e.deferred.resolve() or e.deferred.reject(). You may pass values back through these methods. If you pass a value back inside reject() it will be printed in the console as an error log automatically.
data	object	This is the object containing command options passed in when the command was called. If no options were passed, this will default to an empty object.

Returns

Returns true.

oMyNewPlugin.registerEvents

Registering events is a formality that allows CXBus to keep a registry of all possible events. You don't need to register events before publishing them, but it's a best practice to always register events.

Example

```
oMyNewPlugin.registerEvents(["ready", "testEvent"]);
```

Arguments

Name	Type	Description
Event name array	array	An array of event names.

Returns

Returns true if at least one value event was included in the array. Returns false if no events are included in the array or no array is passed in.

oMyNewPlugin.subscribe

Subscribes your plugin to an event on the bus with a callback function. When the event is published, the callback function is executed. You can subscribe to any event, even if the event does not exist. This allows for binding events that may come in the future.

Example

```
oMyNewPlugin.subscribe("WebChat.opened", function(e){  
    // e = Event Object. Contains metadata and attached data  
    //  
    // Example Event Object data:  
    //  
    // e.time == 1532017560154  
    // e.event == "WebChat.opened"  
    // e.publisher == "WebChat"  
});
```

Arguments

Name	Type	Description
Event name	string	The name of the event you want to subscribe to. Must include the plugin's namespace.
Callback function	function	A function to execute when the event is published. An Event Object is passed into this function that gives you access to metadata and attached data.

Event object

Name	Type	Description
time	number (integer time)	The time the event was published.
event	string	The name of the event, including namespace. That can be useful if you are using the same function to handle multiple events.
publisher	string	The namespace of the plugin that published the event.

Returns

Returns the name of the event back to you if the subscription was successful. Returns false if you did not specify an event and/or a callback function.

oMyNewPlugin.publish

Publishes an event on the bus under your plugin's namespace.

Example

```
// Publishes the event "MyNewPlugin.testEvent" with attached data {test: "123"}
oMyNewPlugin.publish("testEvent", {test: "123"});
```

Arguments

Name	Type	Description
Event name	string	The name of the event you want to publish. Do not include the plugin namespace.
Attached data	object	An object of arbitrary properties you can attach to your event.

Returns

Always returns true.

oMyNewPlugin.republish

A special method of publishing intended for one-off events like "ready". In some cases, an event will fire only once. If a plugin is loaded at a later time that needs to subscribe to this event, it will never get it because it will never be published again. To solve this problem, the "republish" method will automatically republish an event to new subscribers as soon as they subscribe to it.

In Genesys Widgets, every plugin publishes a "ready" event. This event is published using "republish" so that any plugin loaded and/or initialized after can still receive the event.

It is important that you only use "republish" for events that publish once. Using republish multiple

times for the same event can cause unwanted behavior.

Genesys Widgets plugins all publish a "ready" event. This is not related to the CXBus plugin interface object's "ready()" method. Calling `oMyNewPlugin.ready()` will not publish any events.

Example

```
oMyNewPlugin.republish("ready", {...});
```

Arguments

Name	Type	Description
Event name	string	The name of the event you want to have republished. Do not include the plugin namespace.
Attached data	object	An object of arbitrary properties you can attach to your event.

Returns

Always returns true.

`oMyNewPlugin.publishDirect`

A slight variation on "publish", this method will only publish an event on the bus if it has subscribers. The intention of this method is to avoid spamming the logs with events that no plugins are listening to. In particular, if you have an event that publishes frequently or on an interval, "publishDirect" may be used to minimize its impact on logs in the console.

Example

```
oMyNewPlugin.publishDirect("poll", {...});
```

Arguments

Name	Type	Description
Event name	string	The name of the event you want to have republished. Do not include the plugin namespace.
Attached data	object	An object of arbitrary properties you can attach to your event.

Returns

Always returns true.

`oMyNewPlugin.command`

Have your plugin call a command on the bus.

Example

```
oMyNewPlugin.command("WebChat.open", {...}).done(function(e){
    // If command succeeds
    // e == any returned data
}).fail(function(e){
    // If command fails
    // e == any returned data
}).always(function(){
    // Always executed
});
```

Arguments

Name	Type	Description
Command name	string	Name of the command you wish to call.
Command options	string	Optional: An object containing properties the command will use in its execution. Refer to plugin references for a list of options available for each command.

Returns

Always returns a promise. You can define `done()`, `fail()`, or `always()` callbacks for every command.

oMyNewPlugin.before

Allows you to interrupt a registered command on the bus with your own "before" function. You may modify the command options before they're passed to the command, you may trigger some action before the command is executed, or you can cancel the command before it executes.

You may specify more than one "before" function for a command. If you do, they will be executed in a chain where the output of the previous function becomes the input for the next function. You cannot remove "before" functions once they have been added.

Example

```
oMyNewPlugin.before("WebChat.open", function(oData){
    // oData == the options passed into the command call
    // e.g. if this command is called: oMyPlugin.command("WebChat.open", {form: {firstname:
    "Mike"
    // then oData will == {form: {firstname: "Mike"
    // You must return oData back, or an empty object {} for execution to continue.
    // If you return false|undefined|null or don't return anything, execution of the command
    will be stopped
    return oData;
});
```

Arguments

Name	Type	Description
Command name	string	Name of the function you want to interrupt with your "before" function.
"before" function	function	A function that accepts command options (oData in above example). If you want the command to continue executing, you must return the oData object. If you want to cancel the command, return false or undefined or don't return anything. You may modify the contents of oData before it is sent to the command. This allows you to override command options or add on dynamic options depending on external conditions.

Returns

Returns true when you pass a properly formatted command name (e.g. "PluginName.commandName"). Returns false when you pass an improperly formatted command name.

oMyNewPlugin.registry

Returns the CXBus Registry lookup table.

Example

```
oMyNewPlugin.registry();
```

Arguments

No arguments.

Returns

Returns the internal CXBus registry that tracks all plugins, their commands, and their events.
Registry Structure Example:

```
{
  "Plugin1": {
    commands: ["command1", "command2"],
    events: ["event1", "event2"]
  },
  "Plugin2": {
```

```
        commands: ["command1", "command2"],
        events: ["event1", "event2"]
    }
}
```

oMyNewPlugin.subscribers

Returns a list of events and their subscribers.

Example

```
oMyNewPlugin.subscribers();
```

Arguments

No arguments.

Returns

Returns an object identifying a list of events being subscribed to, and a list of plugin names subscribed to each event.

Example of WebChatService's subscribers:

```
// Format  {"eventname": ["subscriber1", "subscriber2"]}

{
    "WebChatService.agentConnected": ["WebChat"],
    "WebChatService.agentDisconnected": ["WebChat"],
    "WebChatService.ready": [],
    "WebChatService.started": ["WebChat"],
    "WebChatService.restored": ["WebChat"],
    "WebChatService.clientDisconnected": [],
    "WebChatService.clientConnected": [],
    "WebChatService.messageReceived": ["WebChat"],
    "WebChatService.error": ["WebChat"],
    "WebChatService.restoreTimeout": ["WebChat"],
    "WebChatService.restoreFailed": ["WebChat"],
    "WebChatService.ended": ["WebChat"],
    "WebChatService.agentTypingStarted": ["WebChat"],
    "WebChatService.agentTypingStopped": ["WebChat"],
    "WebChatService.restoredOffline": ["WebChat"],
    "WebChatService.chatServerWentOffline": ["WebChat"],
    "WebChatService.chatServerBackOnline": ["WebChat"],
    "WebChatService.disconnected": ["WebChat"],
    "WebChatService.reconnected": ["WebChat"]
}
```

oMyNewPlugin.namespace

Returns your plugin's namespace.

Example

```
oMyNewPlugin.namespace();
```

Arguments

No arguments.

Returns

Returns your plugin's namespace. If your plugin's namespace is "MyNewPlugin", it will return "MyNewPlugin".

`oMyNewPlugin.ready`

Marks your plugin as ready to have its commands called. This method is required to be called for all plugins. You should call this method after all your commands are registered, initialization code is finished, and configuration has completed. Failure to call this method will result in your commands being unexecutable.

Example

```
oMyNewPlugin.ready();
```

Arguments

No arguments.

Returns

Returns nothing.

App

Contents

- **1 Overview**
 - 1.1 Usage
 - 1.2 Customization
 - 1.3 Mobile support
- **2 Configuration**
 - 2.1 Description
 - 2.2 Example
 - 2.3 Options
- **3 Localization**
- **4 API commands**
 - 4.1 setTheme
 - 4.2 getTheme
 - 4.3 reTheme
 - 4.4 themeDemo
 - 4.5 setLanguage
 - 4.6 closeAll
 - 4.7 updateAJAXHeader
 - 4.8 removeAJAXHeader
 - 4.9 registerExtension
 - 4.10 registerAutoLoad
 - 4.11 deregisterAutoLoad
- **5 API Events**

- Developer

Learn how to control your widgets.

Related documentation:

-

Overview

App is the main controller for Genesys Widgets and has no UI. It controls all startup routines, global configurations, and extensions, and it executes the `onReady` event and distributes changes to theme, language, mobile mode, and other application-wide effects.

Usage

App's main interface is its configuration. You set all global defaults using the **`window._genesys.widgets.main`** property. App also has a few commands you can use to change the language and theme.

Customization

App itself cannot be customized, but its *configuration options affect all widgets*.

Mobile support

App has built-in mobile detection and can automatically notify all widgets to switch to mobile mode. You can also control this manually.

Configuration

Description

App uses the configuration property **`_genesys.widgets.main`**. App controls the Genesys Widgets product as a whole, handling themes, languages, and mobile devices.

Example

```
window._genesys.widgets = {  
  main: {  
    theme: 'dark',
```

```
themes: {
    dark: 'cx-theme-dark',
    light: 'cx-theme-light',
    blue: 'cx-theme-blue',
    red: 'cx-theme-red'
},
lang: 'en',
il8n: 'il8n.json',
mobileMode: 'auto',
mobileModeBreakpoint: 600,
debug: true,
downloadGoogleFont: true,
googleFontUrl: 'https://apps.mypurecloud.com/webfonts/roboto.css',
header: {'Authorization': 'value'},
cookieOptions: {
    secure: true,
    domain: 'genesys.com',
    path: '/',
    sameSite: 'Strict'
}
},
onReady: function(){
    // Do something on Widgets ready
}
}
```

Options

Name	Type	Description	Default	Required	Introduced/Updated
main.themes	object	An object list containing the CSS classname for each theme. The property names are used to select the theme in the theme property, for example {dark: cx-theme-dark, light: cx-theme-light, red: cx-theme-red, blue: cx-theme-blue}. Where dark and light	{dark: 'cx-theme-dark', light: 'cx-theme-light'}	n/a	

Name	Type	Description	Default	Required	Introduced/ Updated	
		<p>are the built-in themes provided in Genesys Widgets, and red and blue are example custom theme names you may create on your own.</p> <p>Important</p> <p>It is not necessary to define the dark and light theme as shown in this example. It is included to help show how the formatting works. Whatever you put in this object will be merged with the default themes object internally.</p>				
main.theme	string	Selects the theme to apply to Genesys Widgets from the Themes object. Uses the property name of the theme, for example using the example from themes above, possible values for this could be dark, light, red, blue.	dark	n/a		

Name	Type	Description	Default	Required	Introduced/ Updated	
main.lang	string	Select the language to use from the i18n language pack. Language codes are selected by the customer. Any language code format can be used as long as this property matches one of the language codes in your i18n language pack. For more information about localization, see localization.	en	n/a		
main.i18n	URL string or JSON	Either a path to a remote i18n.json language pack file or an inline JSON language pack definition. For more information about language packs, see localization.	en	Default English language strings are built into each widget and are displayed by default. Defining this i18n language pack overrides the built-in strings.	n/a	
main.header	object	An object containing a key value pair for the authorization header.	n/a	n/a	9.0.002.06	
	array	Note: For	none	When lazy		

Name	Type	Description	Default	Required	Introduced/ Updated	
main.preload		<p>use with lazy loading only. A list of plugins you want pre-loaded at startup. You may want certain plugins, such as SideBar, to be shown on screen as soon as possible; to do so, you may add 'sidebar' to this preload plugins array so it will be loaded after Widgets starts up. The names you add to the list must match the first part of the plugin filename you wish to load. Example: sidebar will load sidebar.min.js from the plugins/ folder. All filenames are lowercase.</p> <div> Important This preload array is intended for use when running widgets in lazy loading mode. You may also use this to pre- </div>		loading Widgets		

Name	Type	Description	Default	Required	Introduced/ Updated	
		load your own custom-made plugins.				
main.mobileMode	boolean/ String	Mobile Mode setting. true = Force Mobile Mode on all devices. false = Disable Mobile Mode completely. auto = Genesys Widgets automatically switches between mobile and desktop modes using the mobileModeBreakpoint property and UserAgent detection.	auto	n/a		
main.timeFormat	number/ String	This sets the time format for the timestamps. It can be 12 or 24.	12	n/a		
main.mobileModeBreakpoint	number	The breakpoint width in pixels where Genesys Widgets will switch to Mobile Mode. Breakpoint checked at startup only.	600	n/a		
main.debug	boolean	Enable debug logging from the bus to appear in the browser console.	false	n/a		
main.customStyleSheetID	String	The HTML ID of a	n/a	n/a		
main.downloadGoogleFont	boolean	By default, Genesys	true	n/a		

Name	Type	Description	Default	Required	Introduced/ Updated	
		Widgets downloads and uses the Google font Roboto. To disable this download, set value false.				
main.googleFont	string	<p>The string used to refer the URL where the Google fonts are hosted in Genesys Hosted Repository. You can configure one of the Genesys Hosted region font URLs specified here Genesys Web Fonts.</p> <div> Important This Option is only applicable when the downloadGoog option is set to true. </div>	https://apps.mypurecloud.com/webfonts/roboto.css	n/a	9.0.018.00	
main.deployment	string	The string used to customize cookie names so that multiple Widgets deployments can run in the same domain.	n/a	n/a	9.0.006.02	
main.cookieOptions	object	An object containing cookie attributes that applies	{sameSite:'Strict'	n/a	9.0.017.01	

Name	Type	Description	Default	Required	Introduced/ Updated	
		<p>globally to all Widgets. The following cookie attributes are supported:</p> <ol style="list-style-type: none">1. secure - Either true or false, indicating if the cookie transmission requires a secure protocol (https).2. domain - A string indicating a valid domain where the cookie should be visible.3. path - A string indicating the path where the cookie is visible.4. expires - Specifies the number of days, either from time of creation or from a date instance,				

Name	Type	Description	Default	Required	Introduced/ Updated	
		<p>until the cookie is to be removed. domain and path can be used to make cookies compatible with environments that use a non FQDN URL, such as an intranet hostname. However, the domain should only be manually set in production if the automated values are causing problems. Otherwise, rely on the automated domain and path.</p> <p>5. sameSite - This maps to the cookie SameSite attribute allowing the cookie to</p>				

Name	Type	Description	Default	Required	Introduced/ Updated
		<p>be restricted to a first-party or same-site context. It can take any of the supported values that SameSite attribute takes.</p> <div><h3>Importance</h3><p>The values are automatically set by Widgets to support cross-sub-domain cookies. Modifying these options overrides the automated values and might break cross-sub-domain cookie support if not properly set. For usage, please refer to the above example</p></div>			
onReady	function	A callback function that	none	n/a	

Name	Type	Description	Default	Required	Introduced/ Updated	
		is invoked when the Widgets are ready and initialized with the configuration provided.				

Localization

No localization options.

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('App.themeDemo');
```

setTheme

Sets the theme for Genesys Widgets from the list of registered themes. Default themes are *light* and *dark*. You can register as many new themes as you need.

Example

```
oMyPlugin.command('App.setTheme', {theme: 'light'}).done(function(e){  
    // App set theme successfully  
}).fail(function(e){  
    // App failed to set theme  
});
```


Options

Option	Type	Description
theme	string	Name of the theme you want to use. This name is specified in window.genesys.main.themes . Default themes are light and dark.

Resolutions

Status	When	Returns
resolved	Theme exists and is successfully changed	The name of the theme that was chosen, for example <i>light</i> .
rejected	Theme does not exist	'Invalid theme specified'.

getTheme

Get the CSS classname for the currently selected theme.

Example

```
oMyPlugin.command('App.getTheme').done(function(e){  
    // App got theme successfully  
    // e == CSS classname for current theme  
}).fail(function(e){  
    // App failed to get theme  
});
```

Resolutions

Status	When	Returns
resolved	Always	CSS classname for the currently selected theme, for example, <i>cx-theme-light</i> .
rejected	Never	n/a

reTheme

Accepts an HTML reference (either string or jQuery wrapped set) and applies the proper CSS theme classname to that HTML and returns it back. When widgets receive the **theme** event from App, they pass in their UI containers into App.reTheme to have the old theme classname stripped and the new

classname applied.

Example

```
oMyPlugin.command('App.reTheme', {html: '
Test Theme
'}).done(function(e){
    // App set theme successfully
}).fail(function(e){
    // App failed to set theme
});
```

Options

Option	Type	Description
html	string or jQuery Wrapped Set	HTML string or jQuery Wrapped Set you want to have modified.

Resolutions

Status	When	Returns
resolved	HTML is provided and theme is updated	HTML that was passed-in and modified
rejected	No HTML is provided	'No HTML provided by [plugin name]'

themeDemo

Start an automated demo of each theme. All registered themes will be applied with a default delay between themes of 2 seconds. You can override this delay. This command is useful for comparing themes or testing themes with official or custom widgets.

Example

```
oMyPlugin.command('App.themeDemo', {delay: 1000}).done(function(e){
    // App demo successfully started
}).fail(function(e){
    // App failed to start demo
});
```

Options

Option	Type	Description
delay	number	Number of milliseconds between theme changes. Default value is

Option	Type	Description
		2000 milliseconds.

Resolutions

Status	When	Returns
resolved	Always	n/a
rejected	Never	n/a

setLanguage

Changes the language

Example

```
oMyPlugin.command('App.setLanguage', {lang: 'eng'}).done(function(e){  
    // App set language successfully started  
}).fail(function(e){  
    // App failed to set language  
});
```

Options

Option	Type	Description
lang	string	Change the language of Genesys Widgets. Switches all strings in Widgets to selected language.

Resolutions

Status	When	Returns
resolved	Language successfully changed	n/a
rejected	No language code is provided	No language code provided
rejected	No matching language code is specified in your language pack	No matching language code found in language pack

closeAll

Publishes the 'App.closeAll' event that requests all widgets to close.

Example

```
oMyPlugin.command('App.closeAll').done(function(e){
```

App

```
// App closed all successfully
}).fail(function(e){
    // App failed to close all
});
```

Resolutions

Status	When	Returns
resolved	Always	n/a
rejected	Never	n/a

updateAJAXHeader

Introduced: 9.0.002.06

Updates the Authorization header.

Example

```
_genesys.widgets.bus.command('App.updateAJAXHeader', {header:
    {'Authorization': 'value'}}
);
```

Resolutions

Status	When	Returns
resolved	Header is updated	n/a
rejected	Never	No request header found

removeAJAXHeader

Introduced: 9.0.002.06

Removes the set Authorization header.

Example

```
_genesys.widgets.bus.command('App.removeAJAXHeader');
```

Resolutions

Status	When	Returns
resolved	Always	n/a

registerExtension

Introduced: 9.0.002.06

Allows you to register and initialize new extensions at runtime instead of predefining extensions before Genesys Widgets starts up.

Options

Option	Type	Description
undefined	function	Your extension function. Receives the following arguments: \$ (jQuery), CXBus, Common

Resolutions

Status	When	Returns
resolved	Valid 'extension' object provided	n/a
rejected	Invalid 'extension' option provided	n/a

registerAutoLoad

(For use with lazy loading only) Allows you to register a plugin into the preload plugins array so that it can be pre-loaded at the startup rather than lazy loading later. This can be useful when there is an active session maintained by your Widget and you would like to show it immediately at startup during page refresh or navigating across pages.

Note: This command is intended for use when running widgets in lazy loading mode. You may also use this to register and pre-load your own custom-made plugins.

Options

Option	Type	Description
name	string	The name of the plugin that needs to be registered for auto loading.

Resolutions

Status	When	Returns
resolved	A plugin is added into the preload list	n/a
rejected	Never	n/a

deregisterAutoLoad

(For use with lazy loading only) Allows you to de-register a plugin from the preload plugins array so that it will not be pre-loaded at startup. This can be useful when there is no more active session

maintained by your Widget and you don't want to show it on the screen immediately at startup.

Note: This command is intended for use when running widgets in lazy loading mode. You may also use this to de-register your own custom-made plugins.

Options

Option	Type	Description
name	string	The name of the plugin that needs to be de-registered from auto loading.

Resolutions

Status	When	Returns
resolved	A plugin is removed from the preload list	n/a
rejected	Never	n/a

API Events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('App.ready', function(e){});
```

Name	Description	Data
ready	CallUs is initialized and ready to accept commands.	
i18n	Published when the language for Genesys Widgets is changed or is being set for the first time.	'(language code)'
theme	Published when the theme for Genesys Widgets is changed or is	{theme: '(theme CSS classname)'}

Name	Description	Data
	being set for the first time.	
timeFormat	Published when the time format for Genesys Widgets is changed or is being set for the first time.	{timeFormat: iTimeFormat}

Common

Contents

- [1 Common.Generate.Container\({options}\)](#)
 - [1.1 Example](#)
 - [1.2 Arguments](#)
- [2 Common.Generate.Buttons\({options}\)](#)
 - [2.1 Example](#)
 - [2.2 Arguments](#)
- [3 Common.Generate.Icon\(name\)](#)
 - [3.1 Example](#)
 - [3.2 Arguments](#)
- [4 Common.Generate.Scrollbar\(element, {options}\)](#)
 - [4.1 Example](#)
 - [4.2 Arguments](#)
- [5 Common.config\(object\)](#)
 - [5.1 Example](#)
 - [5.2 Arguments](#)
- [6 Common.checkPath\(object, path\)](#)
 - [6.1 Example](#)
 - [6.2 Arguments](#)
- [7 Common.createPath\(object, path, value\)](#)
 - [7.1 Example](#)
 - [7.2 Arguments](#)
- [8 Common.linkify\(string, options\)](#)
 - [8.1 Example](#)
 - [8.2 Arguments](#)
- [9 Common.log\(mixed, type\)](#)
 - [9.1 Example](#)
 - [9.2 Arguments](#)

- [10 Common.sanitizeHTML\(string\)](#)
 - [10.1 Example](#)
 - [10.2 Arguments](#)
- [11 Common.updateTemplate18n\(element, object\)](#)
 - [11.1 Example](#)
 - [11.2 Arguments](#)
- [12 Common.debugIcons](#)
 - [12.1 Example](#)
- [13 Common.debug](#)
 - [13.1 Example](#)
 - [13.2 Arguments](#)
- [14 Common.error](#)
 - [14.1 Example](#)
 - [14.2 Arguments](#)
- [15 Common.populateAllPlaceholders](#)
 - [15.1 Example](#)
 - [15.2 Arguments](#)
- [16 Common.populateLanguageStrings](#)
 - [16.1 Example](#)
 - [16.2 Arguments](#)
- [17 Common.populateIcons](#)
 - [17.1 Example](#)
 - [17.2 Arguments](#)
- [18 Common.insertIcon](#)
 - [18.1 Example](#)
 - [18.2 Arguments](#)
- [19 Common.injectScript](#)
 - [19.1 Example](#)
 - [19.2 Arguments](#)
- [20 Common.mobileScreenScale](#)
 - [20.1 Example](#)
 - [20.2 Arguments](#)
- [21 Common.showLoading](#)

- [21.1 Example](#)
- [21.2 Arguments](#)
- [22 Common.hideLoading](#)
 - [22.1 Example](#)
 - [22.2 Arguments](#)
- [23 Common.showWaiting](#)
 - [23.1 Example](#)
 - [23.2 Arguments](#)
- [24 Common.hideWaiting](#)
 - [24.1 Example](#)
 - [24.2 Arguments](#)
- [25 Common.watch](#)
 - [25.1 Example](#)
 - [25.2 Arguments](#)
- [26 Common.addDialog](#)
 - [26.1 Example](#)
 - [26.2 Arguments](#)
- [27 Common.showDialog](#)
 - [27.1 Example](#)
 - [27.2 Arguments](#)
- [28 Common.hideDialog](#)
 - [28.1 Example](#)
 - [28.2 Arguments](#)
- [29 Common.hideDialogs](#)
 - [29.1 Example](#)
 - [29.2 Arguments](#)
- [30 Common.showAlert](#)
 - [30.1 Example](#)
 - [30.2 Arguments](#)
- [31 Common.bytesToSize](#)
 - [31.1 Example](#)
 - [31.2 Arguments](#)
- [32 Common.getFormattedTime](#)

- [32.1 Example](#)
- [32.2 Arguments](#)

- Developer

Learn how to access Widgets utility functions and dynamically generate the common HTML containers used throughout Genesys Widgets.

Related documentation:

-

Common is a utility object available for import into Plugins/Widgets and Extensions. It is also accessible directly from the path **window._genesys.widgets.common**.

Common provides utility functions and dynamically generates common HTML Containers used throughout Genesys Widgets.

For all examples below, assume that `_genesys.widgets.common` has been stored in a local variable named 'Common'.

```
var Common = _genesys.widgets.common;
```

Common.Generate.Container({ options })

Dynamically generates a new HTML Container in matching the style of Genesys Widgets with the selected components you request in your options object. Returns the generated container HTML as a jQuery wrapped set.

Example

'Generate an Overlay Container'

```
var ndContainer = Common.Generate.Container({  
    type: 'overlay',  
    title: 'My Overlay', body: 'Some HTML here as a string or jQuery wrapped set',  
    icon: 'call-outgoing',  
    controls: 'close',  
    buttons: false  
}),
```

'Generate a Toast Container'

```
var ndContainer = Common.Generate.Container({  
    type: 'generic',  
    title: 'My Toast', body: 'Some HTML here as a string or jQuery wrapped set',  
    icon: 'chat',  
    buttons: false  
}),
```

```
        controls: '',
        buttons: {
            type: 'binary',
            primary: 'OK',
            secondary: 'cancel'
        }
    }
},
```

Arguments

Argument	Type	Description
options	object	An object containing options to apply to the generated container.
options.type	string	'generic' or 'overlay'. Overlay containers have special CSS properties for appearing inside the Overlay widget. Default is 'generic'.
options.title	string	Title to apply to the container's titlebar area.
options.body	string or jQuery wrapped set	The HTML body you want the container to wrap.
options.icon	string	CSS Classname of icon to use.
options.controls	string	Select from a set of window control buttons to show at the top right. 'close' = Show only the close button. 'minimize' = Show only the minimize button. 'all' = Show both close and minimize buttons.
options.buttons	object	Options for displaying action buttons at the bottom of the container, such as OK and Cancel buttons.
options.buttons.type	string	Currently 'binary' is the only supported button set at this time. Additional sets and arrangements will be available in a later release. Please pass 'binary' as the type here if you wish to show typical 'accept' and 'dismiss' buttons.
options.buttons.primary	string	Display name on the primary button. (for example 'OK', 'Yes', 'Accept', 'Continue', etc.)
options.buttons.secondary	string	Display name on the secondary button. (for example 'Cancel', 'No', 'Dismiss', 'Reject', etc.)

Common.Generate.Buttons({options})

Dynamically generates a new HTML Binary Button set in matching the style of Genesys Widgets with the selected options in your options object. Returns the buttons as a jQuery wrapped set.

Example

'Generate Binary Buttons'

```
var ndButtons = Common.Generate.Buttons({
    type: 'binary',
    primary: 'OK',
    secondary: 'Cancel'
}),
```

Arguments

Argument	Type	Description
options	object	Options for generating buttons, such as OK and Cancel buttons.
options.type	string	Currently 'binary' is the only supported button set at this time. Additional sets and arrangements will be available in a later release. Please pass 'binary' as the type here if you wish to show typical 'accept' and 'dismiss' buttons.
options.primary	string	Display name on the primary button. (for example 'OK', 'Yes', 'Accept', 'Continue', etc.)
options.secondary	string	Display name on the secondary button. (for example 'Cancel', 'No', 'Dismiss', 'Reject', etc.)

Common.Generate.Icon(name)

Dynamically generates an icon from the included icon set. Icons are in SVG format.

Example

'Generate Chat Icon'

Common

```
var ndChatIcon = Common.Generate.Icon('chat');
```

'Insert Chat Icon'

```
$('#your_icon_container').append(Common.Generate.Icon('chat'));
```

Arguments

Argument	Type	Description
name	string	Select the icon you want to generate by name. See the icon reference page for icon names.

Common.Generate.Scrollbar(element, {options})

Dynamically generates a widget scrollbar for selected DOM element.

Example

'Generate Scrollbar for a container'

```
var scrollContainer = Common.Generate.Scrollbar($('#your_container'))
```

Arguments

Argument	Type	Description
element	DOM element or jQuery selector	Select the element to which you would like to apply scrollbar.
options	object	This is an iScroll component. So, all the options that iScroll supports can be passed here.

Common.config(object)

Configure some debug options for Common at runtime.

Example

'Enable full debug logging'

```
Common.config({debug: true, debugTimestamps: true});
```

Arguments

Argument	Type	Description
object	object	Supported options are 'debug' and 'debugTimestamps'. Setting debug to true will enable debug messages created by Common.log(). Setting debugTimestamps to true will add timestamps to the front of each debug message created by Common.log(). Default value for both is false.

Common.checkPath(object, path)

Check for the existence of a sub-property of an object at any depth. Returns the value of that property if found otherwise it returns **undefined**. Useful for checking configuration object paths without having to check each sub-property level individually.

Example

'Check for window._genesys.main'

```
var oMainConfig = false;

if(oMainConfig = Common.checkPath(window, '_genesys.main')){
    //... Utilize oMainConfig
}
```

Arguments

Argument	Type	Description
object	object	An Object you want checked for a particular sub property at any depth.
path	string	The object path in dot notation

Argument	Type	Description
		you wish to search for.

Common.createPath(object, path, value)

Related to checkPath, createPath lets you specify a target object and path string but lets you create the path and set a value for it. This saves you the pain of defining each node in the path individually. All nodes in your path will be created as objects. Your final node, the property you are trying to create, will be whatever value you assign it.

Example

```
'Create window._genesys.main'
```

```
var oMainConfig = false;
```

```
if(oMainConfig = Common.createPath(window, '_genesys.main', {debug:true})){  
    //... Utilize oMainConfig  
}
```

Arguments

Argument	Type	Description
object	object	An object you want to add your new path to.
path	string	The object path in dot notation you wish to create.
value	any	The value you want to assign to the final node (property) in your path.

Common.linkify(string, options)

Search for and convert URLs within a string into HTML links. Returns transformed string.

Example

```
'Check for window._genesys.main'
```

```
var sString = 'Please visit www.genesys.com';  
sString = Common.linkify(sString, {target: 'self'});
```

```
// sString == 'Please visit www.genesys.com
```

Arguments

Argument	Type	Description
string	string	Any string you want to check for URLs and have them converted.
options	object	A list of options to apply to the linkify operation.
options.target	string	Choose the HTML TARGET attribute to apply to the generated links. Default is '_blank'. Set this option to 'self' to apply the target '_self' to the generated links.

Common.log(mixed, type)

Log something to the browser's console. When using `Common.log`, `_genesys.main.debug` must be set to true to see your logs. This allows you to add debug logging to your code without worrying about unwanted debug messages in production. If timestamps are enabled, they will be prefixed to all messages printed through `Common.log`.

Example

'Check the contents of `window._genesys.main`'

```
var Common = _genesys.widgets.common;
Common.log(window._genesys.main);

if(!window._genesys.main){
    Common.log('window._genesys.main is not defined', 'error');
}
```

Arguments

Argument	Type	Description
mixed	Any	Any value or message you'd like to log.
type	string	You can specify the log type, such as 'log', 'debug' and 'error'. Default type is 'log'. Note, if your browser doesn't support the

Argument	Type	Description
		'debug' or 'error' log type, use 'log' instead.

Common.sanitizeHTML(string)

Search for and escape characters within a string. Returns transformed string. Useful for escaping HTML.

Example

```
'Check for window._genesys.main'

var sString = 'Please visit www.genesys.com';
sString = Common.sanitizeHTML(sString);

// sString == 'Please visit <a href=\'www.genesys.com\' target=\'_self\'>www.genesys.com</a>'
```

Arguments

Argument	Type	Description
string	string	Any string you want to be transformed.

Common.updateTemplateI18n(element, object)

Searches through an element's contents for i18n string elements to update with new strings. Used when updating the language in real-time. Works by searching for elements with the CSS classname 'i18n' and reading the custom attribute 'data-message' to match the string name in the language object. See example below.

Example

```
'Check for window._genesys.main'

var ndContainer = $('


Widgets API Reference



51


```

```
// ndContainer ==
```

Accept

Arguments

Argument	Type	Description
element	jQuery wrapped set	Element you want to search within to replace i18n strings.
object	Object of i18n Strings	The list of languages strings you want to update your UI with. This object comes from the App.i18n event or you can define your own custom object inline or using some other system. Object format is a simple name:value pair format. the 'data-message' attribute on your HTML element must match one of these property names to be updated.

Common.debugIcons

Returns the list of all the Icons with their names that Widgets support.

Example

'Fetch and Display list of icons present in Widgets'

```
Common.debugIcons()
```

Common.debug

Adds debug logs in to the browser's console. When using Common.debug, _genesys.main.debug must be set to true to see your logs. This allows you to add debug logging to your code without worrying about unwanted debug messages in production. If timestamps are enabled, they will be prefixed to all messages printed through Common.debug.

Example

'Check the File upload limits in WebChatService'

```
Common.debug(data_server_returned_file_limits);
```

Arguments

Argument	Type	Description
mixed	Any	Any value or message you'd like to add debug log. Note: This is only supported if your browser supports debug log type.

Common.error

Adds error logs in to the browser's console. When using `Common.error`, `_genesys.main.debug` must be set to true to see your logs. This allows you to add error logging to your code without worrying about unwanted error messages in production.

Example

'Logging error messages'

```
Common.error('A widget plugin did not receive the following config: ....');
```

Arguments

Argument	Type	Description
mixed	Any	Any value or message you'd like to add error log. Note: This is only supported if your browser supports error log type.

Common.populateAllPlaceholders

Adds place holder content to the input elements in a form with the given text strings.

Example

'Show placeholders strings in a form'

```
Common.populateAllPlaceholders($('#your_form'), {strings})
```

Arguments

Argument	Type	Description
Form Selector	jQuery DOM selector for a form	Form containing input elements. Note: Input elements should contain i18n class name and data attribute 'data-message-type' with value 'placeholder' for the place holder details to appear.
Key/Value pairs	object	Placeholder messages that needs to be displayed. This is an object with key-value pairs where, key should be equal to the 'data-message' attribute value of an input element and value can be any text that you would like to display.

Common.populateLanguageStrings

Adds the preferred language place holder text to the given input elements in a form.

Example

'Show placeholders strings in a form'

```
Common.populateLanguageStrings($('#your_form'), {strings})
```

Arguments

Argument	Type	Description
Form Selector	jQuery DOM selector for a form	Form containing input elements. Note: Input elements should contain i18n class name and data attribute 'data-message-type' with value 'placeholder' for the place holder details to appear.
Key/Value pairs	object	Placeholder messages that needs to be displayed. This is an object with key-value pairs where, key should be equal to the 'data-message' attribute value of an input element and value can be

Argument	Type	Description
		any text that you would like to display.

Common.populateIcons

Show all the Icons on a Widget.

Example

'Populate all Widget Icons'

```
Common.populateIcons($('#your_container'));
```

Arguments

Argument	Type	Description
element	jQuery DOM selector	Specify the Widget container for which all the Icons have to be displayed.

Common.insertIcon

Adds an icon before the selected element.

Example

'Insert a check mark icon to an element you desire.'

```
Common.insertIcon($('#your_element'), 'alert-checkmark', 'alert')
```

Arguments

Argument	Type	Description
element	jQuery DOM selector	An html element to which Icon is to be displayed.
icon name	string	Name of the Icon that you would

Argument	Type	Description
		like to display. Note: Refer to <code>Common.debugIcons</code> method to find out all the icons names that widgets supports.
icon Aria Name	string	Name for the icon to be read by screen readers.

Common.injectScript

Injects javascript code dynamically into widgets with the help of a script tag.

Example

'Inject your Widget WebChat extension plugin.'

```
Common.injectScript('path/to/LoadWebChat.ext.js')
```

Arguments

Argument	Type	Description
Script file name	string path to JavaScript file	JavaScript file name that needs to be injected into widgets.

Common.mobileScreenScale

Re-sizes and fits Widget to any mobile screen.

Example

'Fit your widget to any mobile screen.'

```
var mobileScaledWidget = Common.mobileScreenScale($('#your_widget'));
```


Arguments

Argument	Type	Description
element	jQuery DOM Selector	Your main Widget wrapper container selector that contains the entire Widget with 'cx-titlebar', 'cx-body', 'cx-footer', 'cx-button-container' and 'cx-message-container' classes in it.

Common.showLoading

Show loading spinner Icon.

Example

'Show loading spinner during an Ajax request'

```
Common.showLoading($('#your_container'))
```

Arguments

Argument	Type	Description
element	jQuery DOM Selector	An html container where loading spinner should appear. This adds a class name 'cx-loading'.

Common.hideLoading

Remove loading spinner Icon.

Example

'Remove loading spinner after the Ajax request'

```
Common.hideLoading($('#your_container'))
```

Arguments

Argument	Type	Description
element	jQuery DOM Selector	An html container which contains the loading spinner.

Common.showWaiting

Show waiting Icon.

Example

'Show waiting Icon when uploading a file.'

```
Common.showWaiting($('#your_container'),'waiting')
```

Arguments

Argument	Type	Description
element	jQuery DOM Selector	An html container where waiting symbol should appear. This adds a class name 'cx-waiting'.
Aria Label	string	The value of the aria-label attribute for the loading screen icon. The default value is 'waiting'

Common.hideWaiting

Remove waiting Icon.

Example

'Remove waiting Icon after file upload is done.'

```
Common.hideWaiting($('#your_container'))
```

Arguments

Argument	Type	Description
element	jQuery DOM Selector	An html container which contains the waiting symbol.

Common.watch

Repeat your function execution for every 'x' milliseconds (default 1 second) up to a maximum number of times (default - infinite) or till your function returns true.

Example

'Make Request Notifications till none are pending.'

```
Common.watch(function(iteration, maxIterations){  
    if(bRequestNotificationsPending){  
        // ..POST Request  
    }  
    return !bRequestNotificationsPending;  
}, 3000, 30)
```

Arguments

Argument	Type	Description
function name	function	The function that you would like to execute. It should return true/false.
frequency	milliseconds	Execute the function for every 'x' milliseconds until it returns true.
limit	number	The maximum number of times function is executed.

Common.addDialog

Create your own dialog box and append it in to the Widget.

Example

'Add a dialog box on your preferred container div'

```
Common.addDialog($('#your_container'), $('#your_dialog_box'), 'my_warning')
```

Arguments

Argument	Type	Description
element	jQuery selector	The parent container that holds the dialog box.
element	jQuery selector	The actual dialog box that you would like to display. This should contain the data-dialog attribute with the value equal to the dialog box name.
name	string	Dialog box name.

Common.showDialog

Show the dialog box that you prefer, using the dialog box name created with Common.addDialog().

Example

'Show the dialog box created using Common.addDialog()'

```
Common.showDialog($('#your_container'), 'your_dialog_box_name');
```

Arguments

Argument	Type	Description
element	jQuery Selector	The parent container which has the Dialog box appended in to it.
name	string	The actual dialog box name.

Common.hideDialog

Hide the dialog box that you showed using Common.showDialog().

Example

'Hide dialog box'

```
Common.hideDialog($('#your_container'), 'your_dialog_box_name');
```

Arguments

Argument	Type	Description
element	jQuery Selector	The parent container which is showing the dialog box.
name	string	The actual dialog box name.

Common.hideDialogs

Hide all the dialog boxes. Dialog box name is not needed here.

Example

'Hide all dialog boxes.'

```
Common.hideDialogs($('#your_container'));
```

Arguments

Argument	Type	Description
element	jQuery Selector	The parent container which is showing all the dialog boxes.

Common.showAlert

Show a native alert dialog box on the Widget you prefer with your own text message. By default, a primary button is added to dismiss the alert dialog.

Example

Show an alert dialog box on the Widget you prefer. But default it adds the dismiss button.

```
Common.showAlert($('.cx-widget.cx-webchat'), {text: 'your alert message', buttonText: 'Ok'})
```

Arguments

Argument	Type	Description
element	jQuery selector	The Widget plugin container that should display the alert dialog. This should be the top level container wrapper holding the Widget.
options	object	The data options containing the text to be shown on the Alert dialog box.
options.text	string	Display text on the Alert dialog box.
options.buttonText	string	Display text on the primary button. (for example 'OK')

Common.bytesToSize

Convert any number in bytes to Kilobytes, Megabytes, Gigabytes and Terabytes.

Example

'bytes to KB, MB, GB or TB.'

```
var fileSize = Common.bytesToSize(parseInt(fileSizeInBytes));
```

Arguments

Argument	Type	Description
bytes	number	Number in bytes size.

Common.getFormattedTime

Returns time in 12 hrs or 24 hrs format from the actual date timestamp. If no timestamp is provided, it uses current time.

Example

'convert date timestamp to return time in 12 hrs format'

```
var formattedTime = Common.getFormattedTime(timestamp, 12);
```

Arguments

Argument	Type	Description
timestamp	Date	JavaScript Date timestamp object.
format	number	Time format with value 12 or 24.

Overlay

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Customization](#)
 - [1.3 Mobile Support](#)
- [2 Configuration](#)
- [3 Localization](#)
- [4 API commands](#)
 - [4.1 open](#)
 - [4.2 close](#)
- [5 API events](#)

- Developer

Learn how to use an overlay window control that widgets can inject their UI into.

Related documentation:

-

Overview

The Overlay plugin provides an overlay window control that widgets can inject their UI into, accepting the HTML UI, placing it inside an overlay control, and displaying the UI onscreen in a uniform overlay window fashion. This prevents individual widgets from managing the overlay themselves. It also means that each widget's UI can be moved between different container types.

Overlay provides these benefits:

- Shows the UI in the center of the window.
- Open and close transition animations.
- No overlapping overlays. Only one at a time. Automatically managed by the Overlay plugin.
- Auto-recenter as the browser window size is changed.
- Automatic application of mobile styles when running in mobile mode.

Usage

Overlay is easy to use; you simply open and close it. When you call `Overlay.open`, you pass in the HTML content you want to show. If you call `Overlay.open` again while an overlay is already open, it will automatically close the previous overlay before showing yours (unless the previous overlay has reserved the overlay to prevent new overlays).

Important

By default, the overlay has no visible styles or content. You must pass in the HTML you want to show inside the Overlay area. Typically you should create an overlay-type container using `Common.Generate.Container`, put your content inside that, then send the whole thing into `Overlay.open`.

Customization

Overlay does not have customization options.

Mobile Support

Overlay automatically applies mobile CSS styles to its outer container to affect the content within the overlay view. It is up to the content inside the overlay view to dynamically change when the Genesys Widgets `.cx-mobile` CSS classname is applied to an outer container.

Configuration

Overlay does not have configuration options.

Localization

Overlay does not have localization options.

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('Overlay.close');
```

open

Opens the provided HTML in an Overlay View. When successful, it returns back the HTML and a custom close event for you to subscribe to. This alerts you when your overlay instance has been closed. You can also make your overlay immutable so that new overlay instances don't close yours. Only your widget can close its overlay when immutable is set to true.

Example

```
oMyPlugin.command('Overlay.open', {
    html: '
Template
',
    immutable: false,
    group: false
}).done(function(e){
    // Overlay opens successfully
}).fail(function(e){
    // Overlay failed to open
});
```

Options

Option	Type	Description
html	string	HTML String template for overlay window.
immutable	boolean	When set to true, overlay cannot be closed by other plugins.
group	string	The name of the overlay window group you want to add a new overlay view into.

Resolutions

Status	When	Returns
resolved	When overlay is successfully opened	{html: , events:

Status	When	Returns												
		<p>, group: } rejected When no html template is passed 'No HTML content was provided. Overlay has ignored your command.' rejected When overlay is already opened 'Overlay view is currently reserved.'</p> <p>close</p> <p>Closes the Overlay UI. Publishes the appropriate custom close event for current overlay being closed.</p> <p>Example</p> <p>Resolutions</p> <table><tr><th>Status</th><th>When</th><th>Returns</th></tr><tr><td>resolved</td><td>When Overlay is successfully closed.</td><td>n/a</td></tr><tr><td>rejected</td><td>When Overlay is already closed.</td><td>'Overlay view is already closed'</td></tr><tr><td>rejected</td><td>When Overlay view is immutable.</td><td>'Overlay view is currently reserved'</td></tr></table>	Status	When	Returns	resolved	When Overlay is successfully closed.	n/a	rejected	When Overlay is already closed.	'Overlay view is already closed'	rejected	When Overlay view is immutable.	'Overlay view is currently reserved'
Status	When	Returns												
resolved	When Overlay is successfully closed.	n/a												
rejected	When Overlay is already closed.	'Overlay view is already closed'												
rejected	When Overlay view is immutable.	'Overlay view is currently reserved'												

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

Overlay

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('Overlay.ready', function(e){});
```

Name	Description	Data
ready	The Overlay plugin is initialized and ready to accept commands	n/a

Toaster

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Namespace](#)
 - [1.3 Customization](#)
 - [1.4 Mobile support](#)
- [2 Configuration](#)
- [3 Localization](#)
- [4 API Commands](#)
 - [4.1 open](#)
 - [4.2 close](#)
- [5 API Events](#)

- Developer

Learn how to use a toast view control into which widgets can inject their UI.

Related documentation:

-

Overview

The Toaster plugin provides a toast view control that widgets can inject their UI into, accepting the HTML UI, placing it inside a toast view, and displaying the UI on-screen at the lower-bottom-right. When it opens, it slides up from the bottom. When it closes, it slides down until it is off-screen.

Toaster provides these benefits:

- Shows UI as a slide-up toast view in the lower-bottom-right of the screen.
- Open and close transition animations.
- No overlapping toasts; only one at a time. Automatically managed by the Toaster plugin.

Usage

Toaster is easy to use - you simply open and close it. When you call `Toaster.open`, you pass in the HTML content you want to show. If you call `Toaster.open` again while a toast is already open, it will automatically close the previous toast before showing yours (unless the previous toast has reserved the view to prevent new toasts).

Namespace

The Toaster plugin has the following namespaces tied to each of the following types.

Type	Namespace
CXBus—API commands & API events	Toaster
CSS	.cx-toaster

Customization

Toaster does not have customization options.

Mobile support

Toaster does not have mobile-specific styles at this time.

Configuration

Toaster does not have configuration options.

Localization

Toaster does not have localization options.

API Commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('Toaster.close');
```

open

Opens the Toaster UI.

Example

```
oMyPlugin.command('Toaster.open', {  
  type: 'generic',  
  title: 'Toaster Title',  
  body: 'Toaster Body',  
  icon: 'chat',  
  controls: 'close',  
  immutable: false,  
  buttons:{  
    type: 'binary',  
    primary: 'Accept',  
    secondary: 'Decline'  
  }  
})
```


Toaster

```
}).done(function(e){  
    // Toaster opened successfully  
}).fail(function(e){  
    // Toaster failed to open properly  
});
```

Options

Option	Type	Description
type	string	Specifies the type of body content that can be provided to Toaster window. Generic type shows the default body content and custom type overrides the default html body content.
title	string	Heading title to display on the Toaster window.
body	string	Holds text value for Generic Toaster type and html string template for Custom Toaster type.
icon	string	The CSS class name for an icon.
controls	string	Show close and minimize controls on Toaster window.
buttons	object	Define the type of buttons.
buttons.type	string	Shows two buttons on the Toaster .
buttons.primary	string	Text to be shown on primary button.
buttons.secondary	string	Text to be shown on secondary button.
immutable	boolean	When set to true, Toaster cannot be closed by other plugins.

Resolutions

Status	When	Returns
resolved	Toaster is successfully opened	n/a
rejected	No Toaster type is specified	'No content was provided. Toaster has ignored your command'
rejected	Toaster is already opened	'Toaster view is currently reserved'

close

Closes the Toaster UI.

Example

```
oMyPlugin.command('Toaster.close').done(function(e){
    // Toaster closed successfully
}).fail(function(e){
    // Toaster failed to close
});
```

Resolutions

Status	When	Returns
resolved	Toaster is successfully closed	n/a
rejected	Toaster is already closed	'Toaster view is already closed'
rejected	Toaster view is immutable	'Toaster view is currently reserved'

API Events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');
oMyPlugin.subscribe('Toaster.ready', function(e){});
```

Name	Description	Data
ready	The Toaster plugin is initialized and ready to accept commands	n/a
closed	The Toaster plugin has been removed from the screen	n/a

WindowManager

Contents

- **1 Overview**
 - **1.1 Usage**
 - **1.2 Customization**
 - **1.3 Screenshot**
- **2 Configuration**
- **3 Localization**
- **4 API Commands**
 - **4.1 registerDockView**
 - **4.2 registerSideButton**
- **5 API Events**

- Developer

Learn how to use the WindowManager plugin, which provides a controller for several different types of window group.

Related documentation:

-

Overview

The WindowManager plugin provides a controller for several different types of window group. HTML UIs added to these WindowManager groups are arranged and managed in accordance with each group's purpose.

One group type is *Dock View*, which appears as a toast-like UI docked in the lower-bottom-right of the screen. This group automatically stacks widgets horizontally. When one of the widgets closes, the stack collapses toward the right. Widgets can register themselves into this WindowManager group and let it do all the work.

Another group type is *Side Button*, with its launcher button on the right side of the screen. Like the dock view, buttons are stacked, but in this case they are stacked vertically. As buttons are added and removed from the group, the button stack collapses to fill in the gaps.

Usage

WindowManager has "register" commands for registering your UI into different groups. They all accept one argument, the HTML you want to be handled by WindowManager. You can use 'registerDockView' or 'registerSideButton' at this time. More window management groups will be added in upcoming releases.

Customization

WindowManager does not have customization options.

Screenshot



Configuration

WindowManager does not have configuration options.

Localization

WindowManager does not have localization options.

API Commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');

oMyPlugin.command('WindowManager.registerDockView', {html: '
HTML
'});
```

registerDockView

Creates a docked view container to show a widget on the bottom right corner. Its position is adjusted (stacked) to appear beside another widget if already present and is indexed with a tabindex.

Example

```
oMyPlugin.command('WindowManager.registerDockView', {html: '
Template
'}).done(function(e){

    // WindowManager registered a dockView successfully

}).fail(function(e){

    // WindowManager failed to register a dock view

});
```

Options

Option	Type	Description
html	string	A Widget HTML string template that needs to be shown in dock view.

Resolutions

Status	When	Returns
resolved	The HTML template is successfully opened and registered in dock view.	n/a
rejected	No HTML template is found.	'No html content'

registerSideButton

Registers a button to show on the right side of the screen for a particular plugin. Its position is based on the respective plugin order defined in the array configuration. Currently, this is not supported for external plugins.

Example

```
oMyPlugin.command('WindowManager.registerSideButton', {template: '
Button Text
'}).done(function(e){
    // WindowManager registered a side button successfully
}).fail(function(e){
    // WindowManager failed to register a side button
});
```

Options

Option	Type	Description
template	string	Custom HTML string template for a button.

Resolutions

Status	When	Returns
resolved	The HTML button is successfully registered.	n/a

Status	When	Returns
rejected	No HTML template is found.	'No button template found to register'

API Events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('WindowManager.ready', function(e){});
```

Name	Description	Data
ready	WindowManager is initialized and ready to accept commands.	n/a
changed	WindowManager publishes this event when there is any change in the position of widgets on the screen.	{registry: (object)}

CallbackService

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Namespace](#)
 - [1.3 Customization](#)
- [2 Configuration](#)
 - [2.1 Description](#)
 - [2.2 Example](#)
 - [2.3 Options](#)
- [3 Localization](#)
- [4 API Commands](#)
 - [4.1 configure](#)
 - [4.2 schedule](#)
 - [4.3 availability](#)
- [5 API Events](#)

- Developer

Learn how to use CallbackService to schedule a callback with customer service.

Related documentation:

-

Overview

CallbackService exposes high-level API access to Genesys Callback services, allowing you to use our Callback Widget to schedule a callback with customer service—or to develop your own custom Callback Widget. CallbackService dramatically simplifies integration, improving the reliability, feature set, and compatibility of every widget on the bus.

Usage

Callback Service and the matching Callback widget work together, and they share a configuration object. Using Callback uses CallbackService.

You can also use Callback Service as a high-level API using bus commands and events to build your own Callback widget.

Namespace

The CallbackService plugin has the following namespaces tied to each of the following types:

Type	Namespace
Configuration	Sendmessage
CXBus—API commands & API events	CallbackService

Customization

CallbackService does not have customization options. It is a Plug and Play plugin and works as is.

Configuration

Description

Callback and CallbackService share the **_genesys.widgets.callback** configuration namespace. Callback contains the UI options and CallbackService contains the connection options.

Example

```
window._genesys.widgets.callback = {
  apiKey: 'n3eNkgXXXXXXXXX0XXXXXXXXXA',
  apiVersion: 'v3',
  serviceName: 'service',
  dataURL: 'http://host:port/callbacks',
  userData: {},
  countryCodes: true
};
```

Options

Name	Type	Description	Accepted Values	Default	Required
apiKey	string	If apiVersion is v3, this holds the x-api-key value.	n/a	n/a	Yes, if using Apigee Proxy
dataURL	URL String	URL to the API endpoint for Callback. Important The base URL for your API endpoints is: https://gapi-.genesysengagement/v3 You will receive the information from Genesys at the same time that you receive your API key.	n/a	n/a	Always
apiVersion	string	Version of Callback API. Important This value determines the version of Callback API.	'v3'	'v1'	Yes, if using Callback v3 dataURL
serviceName	string	Name of the Callback virtual queue.	n/a	n/a	Yes, if using Callback v3 dataURL
userData	object	Arbitrary	n/a	{}	

Name	Type	Description	Accepted Values	Default	Required
		attached data to include while scheduling a callback.			
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout.	n/a	3000	

Localization

CallbackService does not have localization options.

API Commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');
oMyPlugin.command('CallbackService.schedule', {
    userData: {},
    firstname: 'Bob',
    lastname: 'Jones',
    email: 'b.jones@mail.com',
    subject: 'product questions',
    desiredTime: '2017-04-04T00:24:17.804Z',
    phonenumber: '4151110000'
});
```

configure

Internal use only. The main App plugin shares configuration settings with widgets using each widget's configure command. The configure command can only be called at startup. Calling configure again after startup may result in unpredictable behavior.

schedule

Schedule a callback service with the callback schedule API.

Example

```
oMyPlugin.command('CallbackService.schedule', {  
    userData: {},  
    serviceName: 'service' // service name from callback API v3 version,  
    firstname: 'Bob',  
    lastname: 'Jones',  
    email: 'b.jones@mail.com',  
    subject: 'product questions',  
    desiredTime: '2017-03-03T00:24:17.804Z',  
    phonenumber: '4151110000'  
});
```

Options

Option	Type	Description
firstname	string	Receive a Call entry Form Data: 'firstname'.
lastname	string	Receive a Call entry Form Data: 'lastname'.
phonenumber	string	Receive a Call entry Form Data: 'phonenumber'.
subject	string	Receive a Call entry Form Data: 'notes'.
email	string	Receive a Call entry Form Data: 'email'.
desiredtime	string	The preferred desired time user would like to get the callback scheduled. Time should be in UTC format.
userData	object	Arbitrary data that is to be attached with callback schedule. Properties defined here will be merged with default userData set in the configuration object.
serviceName	string	Service Name of Callback API to be passed if the apiVersion is v3.

Resolutions

Status	When	Returns
resolved	Server confirms callback is scheduled	200 OK AJAX Response - Schedule Callback

Status	When	Returns
		For Callback API v3, refer to 'Responses' in Schedule Callback V3
rejected	Selected time slot is not available	400 Bad Request AJAX Error Response For Callback API v3, refer to 'Responses' in Schedule Callback V3
rejected	AJAX exception occurs	429 Too Many Requests AJAX Error Response For Callback API v3, refer to 'Responses' in Schedule Callback V3
rejected	Server exception occurs	500 Internal Server Error Response For Callback API v3, refer to 'Responses' in Schedule Callback V3
rejected	No form data is found to schedule callback	'No data found to schedule callback'

availability

Get the list of available callback time slots using the callback service.

Example

```
oMyPlugin.command('CallbackService.availability', {
    serviceName: 'service' // service name from callback API v3 version,
    startDate: '2017-04-03T00:24:17.804Z',
    numberOfDays: '5',
    maxTimeSlots: 20
}).done(function(e){
    // CallbackService successfully showing availability
}).fail(function(e){
    // CallbackService failed to show availability
});
```

Options

Option	Type	Description
startDate	string	The start date is specified in ISO 8601 format, using UTC as the timezone (yyyy-MM-ddTHH:mm:ss.SSSZ).
endDate	string	The end date is specified in ISO 8601 format, using UTC as timezone (yyyy-MM-ddTHH:mm:ss.SSSZ). If neither endDate nor numberOfDays is specified, the end date is assumed to be the same as the start date.
numberOfDays	string	Used as an alternative to the end date. If neither endDate nor numberOfDays is specified, the end date is assumed to be the same as the start date.
maxTimeSlots	number	The maximum number of time slots to be included in the response.
serviceName	string	Service Name of Callback API to be passed if the apiVersion is v3.

Resolutions

Status	When	Returns
resolved	Server confirms the list of available callback time slots	200 OK AJAX Response - Query Callback Availability For Callback API v3, refer to 'Responses' in Availability Callback V3
rejected	Time slots are not available for selected period	400 Bad Request AJAX Response For Callback API v3, refer to 'Responses' in Availability Callback V3
rejected	AJAX exception occurs	400 Bad Request AJAX Response For Callback API v3, refer to 'Responses' in Availability Callback V3
rejected	Server exception occurs	500 Internal Server Error Response

Status	When	Returns
		For Callback API v3, refer to 'Responses' in Availability Callback V3
rejected	No query data is found	'No query parameters passed for callback availability service'

API Events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');
oMyPlugin.subscribe('CallbackService.ready', function(e){});
```

Name	Description	Data
ready	CallbackService is initialized and ready to accept commands.	n/a
scheduled	Callback is scheduled successfully.	200 OK AJAX Response - Schedule Callback For Callback API v3, refer to 'Responses' in Schedule Callback V3
scheduleError	An error occurred between the client and the server during a callback schedule.	The JSON data returned by Callback server. For Callback API v3, refer to 'Responses' in Schedule Callback V3
availableSlots	Callback available slots fetched successfully.	200 OK AJAX Response - Query Callback Availability For Callback API v3, refer to 'Responses' in Availability Callback V3
availabilityError	An error occurred between the	The JSON data returned by

Name	Description	Data
	client and the server while fetching the available timeslots.	Callback server. For Callback API v3, refer to 'Responses' in Availability Callback V3

StatsService

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Namespace](#)
 - [1.3 Customization](#)
- [2 Configuration](#)
 - [2.1 Description](#)
 - [2.2 Example](#)
 - [2.3 Options](#)
- [3 Localization](#)
- [4 API commands](#)
 - [4.1 configure](#)
 - [4.2 getStats](#)
- [5 API events](#)
- [6 Estimated wait time](#)
 - [6.1 API versions](#)
 - [6.2 Where to look for EWT data](#)

- Developer

Learn how to fetch estimated wait time (EWT) details for a channel.

Related documentation:

-

Overview

StatsService exposes high-level API access to Genesys statistics services, allowing you to fetch estimated wait time (EWT) details for each channel, such as WebChat or Callback, and display these details across the channels.

Usage

StatsService and the Channel Selector widget work together right out of the box to display EWT details across all channels. Using the Channel Selector widget uses StatsService.

You can also use StatsService as a high-level API with bus commands and events and integrate in your own widget.

Namespace

The StatsService plugin has the following namespaces, tied to each of the following types:

Type	Namespace
Configuration	stats
CXBus—API commands & API events	StatsService

Customization

StatsService doesn't have any customization options. It is a plug-and-play plugin and works as is.

Configuration

Description

StatsService shares the **_genesys.widgets.stats** configuration namespace and has connection settings to fetch EWT details from each channel.

Example

```
window._genesys.widgets.stats =
ajaxTimeout: 3000,
ewt: {
  dataURL: 'http://10.0.0.121:7777/genesys/1/service/ewt-for-vq',
  apikey: 'n3exxxxxXREBMjGxxxx8VA',
  apiVersion: 'v1',
  mode: 'urs2'}
};
```

Options

Name	Type	Description	Default	Required	Accepted Values
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout	3000	N/A	N/A
ewt.apikey	string	Apigee Proxy secure token. If apiVersion is v3, this holds the x-api-key value.	N/A	Yes, if using Apigee Proxy or v3 API.	N/A
ewt.dataURL	URL String	URL to the API endpoint for estimated wait time (EWT)	N/A	Always	N/A
ewt.apiVersion	string	Version of EWT API. Note: This value determines the version of EWT API in GMS/v3. That is: 'v1' - GMS EWT v1 'v2' - GMS EWT v2 'v3' - EWT v3 Only GET request type with virtual queue name as query parameters	v1'	Yes, if using GMS EWT v2 or EWT v3 dataURL	'v1', 'v2', 'v3'

Name	Type	Description	Default	Required	Accepted Values
		are supported.			
ewt.mode	string	EWT mode parameter for GMS/v3 API. This value will vary based on the above apiVersion.	Will vary based on the above apiVersion as shown below. 'urs2' for 'v1' 'ewt2' for 'v2' 'mode2' for 'v3'	N/A	'urs', 'urs2' or 'stat' for 'v1' 'ewt1', 'ewt2' or 'ewt3' for 'v2' 'mode1', 'mode2' or 'mode3' for 'v3'

Localization

StatsService doesn't have any localization options.

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('StatsService.getStats');
```

configure

This is for internal use only. The main App plugin shares configuration settings to widgets using each widget's configure command. The configure command can only be called once at startup. Calling configure again after startup may result in unpredictable behavior.

Example

```
oMyPlugin.command('StatsService.configure', {
  ewt:{
    apikey: '12345',
    dataURL: 'http://localhost:8080/foo/bar'
  },
  ajaxTimeout: 10000
}).done(function(e){
  // StatsService configured successfully
}).fail(function(e){
  // StatsService failed to configure
});
```

Options

Option	Type	Description
ewt.apikey	string	API access token. Please contact your Genesys representative to obtain your API access token.
ewt.dataURL	URL String	URL of GES server.
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout.

Resolutions

Status	When	Returns
resolved	Configuration options are provided and set.	n/a
rejected	No configuration options are provided.	'Invalid configuration'

getStats

Ask the Genesys Stat server to fetch EWT details.

Example

```
oMyPlugin.command('StatsService.getStats', {
  group: 'EWT',
  vqName: 'chat_ewt_test_eservices',
  mode: 'urs2'
```

```
}).done(function(e){  
    // StatsService got stats successfully  
}).fail(function(e){  
    // StatsService failed to get stats  
});
```

Options

Option	Type	Description
group	string	Mention specific group name you would like to request, such as EWT, etc.
vqname	string/array	Specify a single virtual queue name as a string or a list of virtual queue names as an array. EWT will be fetched only for these virtual queues specified here. If nothing is specified, EWT will be fetched for all the available virtual queues.
mode	string	Specify EWT mode. This will vary based on apiVersion. Refer to mode configuration option for possible values.

Resolutions

Status	When	Returns
resolved	Server returns EWT data.	(AJAX Response Object)
rejected	Server fail request fails.	'EWT request failed due to unknown reason'
rejected	No EWT dataURL provided.	'Invalid EWT configuration'

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see

Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('StatsService.ready', function(e){});
```

Name	Description	Data
ready	StatsService is initialized and ready to accept commands.	n/a
updated	Latest Stats data is available.	EWT AJAX Response data
error.ewt	An error occurred between the client and the server for EWT.	{(AJAX data Response)}

Estimated wait time

Estimated wait time (EWT) is displayed in the ChannelSelector and Callback widgets. These widgets use the `getStats` command to fetch EWT data from the GMS or GES server. These servers support multiple API versions and this document will explain how to configure the StatsService plugin to use the version that you need.

Use the `ewt.apiVersion` configuration option to specify the API version. Each version value corresponds to a particular API of GMS/GES. For all possible version values and their mapping, refer to the Description section of the `ewt.apiVersion` configuration option.

Sample configuration:

```
_genesys.widgets.stats.ewt.apiVersion =
```

API versions

v1

If `ewt.apiVersion` is configured to **v1** (this is also the default value), the `ewt.dataURL` configured must be a valid GMS 8.5.1 EWT API url. If not, an incorrect EWT value might be displayed.

Depending on this API version, the `ewt.mode` configuration option can hold a set of predefined possible values for this version. They are 'urs', 'urs2' and 'stat', where 'urs2' is the default value if not specified.

Default example

```
_genesys.widgets.stats = {  
  ewt: {  
    apiVersion: "v1"  
    dataURL: http://somedomain/genesys/1/service/ewt-for-vq  
    mode: "urs2"  
  }  
}
```

For the above configuration, the StatsService plugin will construct the relevant dataURL as shown below.

`http://somedomain/genesys/1/service/ewt-for-vq?name=vq1&aqt=urs2`

'vq1' is added to the URL via the **vqName** option passed into the `getStats` command.

v2

If `ewt.apiVersion` is configured to **v2**, the `ewt.dataURL` configured must be a valid GMS 8.5.2 EWT API url. If not, incorrect EWT may be displayed. For this `apiVersion`, the possible values for `ewt.mode` are 'ewt1', 'ewt2' and 'ewt3'. 'ewt2' is the default value.

Example

```
_genesys.widgets.stats = {
  ewt: {
    apiVersion: "v2"
    dataURL: http://somedomain/genesys/2/ewt
    mode: "ewt2"
  }
}
```

For the above configuration, the StatsService plugin will construct the relevant dataURL as shown below.

`http://somedomain/genesys/2/ewt/ewt2?vq=vq1,vq2`

'vq1' and 'vq2' are added to the URL via the **vqName** option passed into the `getStats` command.

v3

If `ewt.apiVersion` is set to **v3**, the `ewt.dataURL` configured must be a valid GES EWT API url. If not, incorrect EWT may be displayed. For this `apiVersion`, the possible values for `ewt.mode` are 'mode1', 'mode2' and 'mode3', where 'mode2' will be the default value if not specified.

Example

```
_genesys.widgets.stats = {
  ewt: {
    apiVersion: "v3"
    dataURL: http://somedomain/engagement/v3/estimated-wait-time
    mode: "mode2"
  }
}
```

For the above configuration, the StatsService plugin will construct the relevant dataURL as shown below.

`http://somedomain/engagement/v3/estimated-wait-time?virtual-queues=vq1,vq2&mode=mode2`

'vq1' and 'vq2' are added to the URL via the **vqName** option passed into the `getStats` command.

Where to look for EWT data

When the `getStats` command is called, it fetches the EWT data from either GMS/GES server based on

the configuration. This response data is included in the updated event in a standard format as shown below. In this data format, the **ewt** section will contain the virtual queue name and the estimated wait time as a key value pair. The **response** section contains the original raw data from the server and may vary between each server API.

```
{
  ewt: {
    "VQ_GMS_Callback_Out": 9.999 // consolidated standardized EWT data for each
virtual queue.
    "VQ_GMS_Callback": 5.12
    ...
  },
  response: { // Original raw data from GMS.
    "VQ_GMS_Callback_Out": {
      "time": 1506021728,
      "wt": 0,
      "calls": 0,
      "wcalls": 0,
      "pos": 1,
      "wpos": 1,
      "aqt": 9.999,
      "ewt": 9.999,
      "hit": 0
    },
    "VQ_GMS_Callback": {
      ...
    }
  }
}
```

WebChatService

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Namespace](#)
 - [1.3 Customization](#)
 - [1.4 Limitations](#)
- [2 Configuration](#)
 - [2.1 Example](#)
 - [2.2 Options](#)
- [3 Localization](#)
- [4 API commands](#)
 - [4.1 configure](#)
 - [4.2 startChat](#)
 - [4.3 endChat](#)
 - [4.4 sendMessage](#)
 - [4.5 sendCustomNotice](#)
 - [4.6 sendTyping](#)
 - [4.7 sendFilteredMessage](#)
 - [4.8 addPrefilter](#)
 - [4.9 updateUserData](#)
 - [4.10 poll](#)
 - [4.11 startPoll](#)
 - [4.12 stopPoll](#)
 - [4.13 resetPollExceptions](#)
 - [4.14 restore](#)
 - [4.15 getTranscript](#)
 - [4.16 getAgents](#)
 - [4.17 getStats](#)
 - [4.18 sendFile](#)

- [4.19 downloadFile](#)
- [4.20 getSessionData](#)
- [4.21 fetchHistory](#)
- [4.22 registerTypingPreviewInput](#)
- [4.23 registerPreProcessor](#)
- [4.24 verifySession](#)
- [5 API events](#)

- Developer

Learn how to use Genesys chat services.

Related documentation:

-

Overview

WebChatService exposes high-level API access to Genesys chat services, so you can monitor and modify a chat session on the front end, or develop your own custom WebChat widgets. Compared to developing a custom chat UI and using the chat REST API, WebChatService dramatically simplifies integration—improving the reliability, feature set, and compatibility of every widget on the bus.

Usage

WebChatService and the matching WebChat widget work together right out of the box and they share the same configuration object. Using WebChat uses WebChatService.

You can also use WebChatService as a high-level API using bus commands and events to build your own WebChat widget or other UI features based on WebChatService events.

Namespace

The WebChat Service plugin has the following namespaces tied to each of the following types:

Type	Namespace
Configuration	webchat
CXBus—API commands & API events	WebChatService

Customization

WebChatService has many configuration options but no customization options. It is a plug-and-play plugin and works as is.

Limitations

Multiple instances of the same chat session

After starting a chat session, that session can be opened in any number of new tabs on the same site. Each tab runs an independent instance of WebChat connected to the same chat session. Currently, Instances are not synchronized with each other due to Nexus limitation.

Configuration

WebChat and WebChatService share the **_genesys.widgets.webchat** configuration namespace. WebChat contains the UI options and WebChatService contains the connection options.

Important

Starting with version 9.0.008.04, WebChatService allows you to choose between the types of chat services available in Genesys via the transport section in configuration options.

Example

// When using v2 API

```
window._genesys.widgets.webchat = {  
  apikey: 'n3eNkgxxxxxxxxxxxx8VA',  
  dataURL: 'https://api.genesyscloud.com/gms-chat/2/chat',  
  enableCustomHeader: true,  
  
  userData: {},  
  emojis: true,  
  actionsMenu: true,  
  
  autoInvite: {  
    enabled: false,  
    timeToInviteSeconds: 10,  
    inviteTimeoutSeconds: 30  
  },  
  
  chatButton: {  
    enabled: true,  
    template: '  
CHAT NOW  
'  
    effect: 'fade',  
    openDelay: 1000,  
    effectDuration: 300,  
    hideDuringInvite: true  
  }  
};
```

// When using v3 API

```
window._genesys.widgets.webchat = {  
  emojis: true,  
  userData: {},  
  transport: {  
    type: 'pureengage-v3-rest',  
    dataURL: 'https://nexus/v3/chat/sessions',  
    endpoint: 'xxxxxxxx',  
  }  
};
```

```

headers: {
  'x-api-key': 'xxxxxxx'
},
async: {
  enabled: true,

  getSessionData: function(sessionData, Cookie, CookieOptions) {
    // Note: You don't have to use cookies. You can, instead,
    store in a secured location like a database.
    Cookie.set('customer-defined-session-cookie',
JSON.stringify(sessionData), CookieOptions);
  },

  setSessionData: function(Open, Cookie, CookieOptions) {
    // Retrieve from your secured location.
    return Cookie.get('customer-defined-session-cookie');
  }
},
chatButton: {
  enabled: true,
  template: '
CHAT NOW
',
  effect: 'fade',
  openDelay: 1000,
  effectDuration: 300,
  hideDuringInvite: true
}
};

```

Options

Version 2 API

Name	Type	Description	Default	Required	Introduced/ Updated
apikey	string	Apigee Proxy secure token. <div> Important This option is only supported in GMS REST mode. </div>	n/a	Yes, if using Apigee Proxy	
endpoint	string	Manually select the endpoint on which to initiate chat.	n/a	n/a	
dataURL	string (URL)	URL for GMS REST chat service. If cometD.enabled	n/a	Always	

Name	Type	Description	Default	Required	Introduced/ Updated
		is set to true, this property will be ignored.			
enableCustomHeader	boolean	Enables the use of the custom authorization header defined in <code>_genesys.widgets.main.header</code> static config. Attaches the custom authorization header to all WebChatService request.	false	No	9.0.002.06
userData	object	Arbitrary attached data to include when initiating a chat.	{}	n/a	
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout.	3000	n/a	
xhrFields	object	Allows you to set the properties for the AJAX <code>xhrFields</code> object (for example, <code>{withCredentials: false}</code>). Important This option is only supported in GMS REST mode.	<code>{withCredentials: false}</code>	n/a	
pollExceptionLimit	number	Number of successive poll exceptions (chat server offline) before WebChatService publishes 'chatServerWentOffline'.	5	n/a	
restoreTimeout	number	Number of milliseconds before restore	60000		

Name	Type	Description	Default	Required	Introduced/ Updated
		timeout. Prevents the chat session from restoring after a certain time away from the session (for example, user navigated to a different site during chat and never ended the session).			

Version 3 API

Name	Type	Description	Default	Required	Introduced/ Updated
transport	object	Object containing the transport service configuration options.	n/a	Yes, when using new transport services available with WebChat.	9.0.008.04
transport.type	string	Select the type of transport service that needs to work with WebChat UI plugin. For Pure Engage v3 REST API, the value is 'pureengage-v3-rest'.	n/a	Yes, when using Pure Engage v3 REST API.	9.0.008.04
transport.dataURLstring (URL)		URL for Pure Engage v3 REST API chat service. Please contact your local Genesys customer representative to obtain a valid dataURL.	n/a	Always	9.0.008.04
transport.endpoint	string	The endpoint for Genesys Multicloud CX v3 API.	n/a	Yes	9.0.008.04
transport.headers	object	Object	n/a	Yes	9.0.008.04

Name	Type	Description	Default	Required	Introduced/ Updated
		containing key value pairs of any custom headers.			
transport.headers[x-api-key]	string	The API key provided from Genesys. Please contact your local Genesys customer representative to obtain a valid API key.	n/a	Yes	9.0.008.04
transport.async	object	<p>Object containing Async mode configuration options.</p> <div> Important To properly restore a chat session that has ended previously, you'll need to navigate back to the page and open the WebChat Widget. This way, the chat session is restored in the background and is ready. Presently, this is a current limitation in Async WebChat. </div>	{ }	No	9.0.008.04
transport.async.enabled	boolean	Enable Asynchronous Chat where a chat session can be active indefinitely. When you close WebChat without ending the chat session, the session will simply go dormant. When you open	false	Yes, when Async WebChat mode is enabled	9.0.008.04

Name	Type	Description	Default	Required	Introduced/ Updated
		WebChat again, the session will restore and continue chatting where left off.			
transport.async.getSessionData	Function	A function that you can define to retrieve updated session data from the WebChatService plugin. This function is called back when starting a new Async chat session for the first time, or when the sessionData changes over the course of an active chat session. This function takes the following arguments: sessionData (current active session data), Cookie (Widgets Internal cookie reference), and CookieOptions (a parameter that is needed when using Widgets Cookie). The purpose of this function is to provide you with the active session data so that it can be stored somewhere safe and secure. Later	none	Yes, when Async WebChat mode is enabled	9.0.008.04

Name	Type	Description	Default	Required	Introduced/ Updated
		this needs to be provided in the below setSessionData function to restore the chat session. Refer to the example for usage.			
transport.async.setSessionData	function	A function that you can define to return the session data to the WebChatService plugin. During initialization, the WebChatService plugin will call this function to check if any session data is returned. If found, WebChatService tries to restore the chat session using this session data and open the WebChat Widget. WebChatService will also pass the following arguments into this function: Open (WebChat current open state value), Cookie (Widgets Internal cookie reference), and CookieOptions (a parameter that is needed when using Widgets Cookie). Refer to the example	none	Yes, when Async WebChat mode is enabled	9.0.008.04

Name	Type	Description	Default	Required	Introduced/ Updated
		for usage.			
transport.async.deleteSessionData	function	A function that you can define to delete the session data from your secret storage, it will be called by WebChatService plugin when Async chat session is lost or cannot find anymore due to unknown reasons. This function will enable you write the script for deleting the session data from your secret storage, in this way WebChat will try to start a new chat normally rather than trying to restore a lost chat session. WebChatService will also pass the following arguments into this function - errorData (lost session and error details), Cookie (Widgets Internal cookie reference) and CookieOptions (a parameter that will be needed when using Widgets Cookie).	none	Yes, when Async WebChat mode is enabled	9.0.015.12
userData	object	Arbitrary attached data to include when initiating	{}	n/a	

Name	Type	Description	Default	Required	Introduced/ Updated
		a chat.			
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout.	3000	n/a	

Localization

WebChatService doesn't have any localization options.

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('WebChatService.getAgents');
```

Important

Starting with version 9.0.008.04, WebChatService allows you to choose between the types of chat API services available in Genesys via the transport section configuration options. For more information, see the Options table in configuration options.

configure

Internal use only. The main App plugin shares configuration settings to widgets using each widget's configure command. The configure command can only be called once at startup. Calling configure again after startup may result in unpredictable behavior.

startChat

Initiates a new chat session with the chat server via GES or with the service configured under the transport section.

Important

The options data must be under the “form” object when using the “WebChatService.startChat” command in v3 API.

Example

```
// When using v2 API
oMyPlugin.command('WebChatService.startChat', {
    nickname: 'Jonny',
    firstname: 'Johnathan',
    lastname: 'Smith',
    email: 'jon.smith@mail.com',
    subject: 'product questions',
    userData: {}
}).done(function(e){
    // WebChatService started a chat successfully
}).fail(function(e){
    // WebChatService failed to start chat
});

// When using v3 API
oMyPlugin.command('WebChatService.startChat', {
    form:{
        nickname: 'Jonny',
        firstname: 'Johnathan',
        lastname: 'Smith',
        email: 'jon.smith@mail.com',
        subject: 'product questions',
    },
    userData: {}
}).done(function(e){
    // WebChatService started a chat successfully
}).fail(function(e){
    // WebChatService failed to start chat
});
```

Options

Option	Type	Description
nickname	string	Chat Entry Form Data: 'nickname'.
firstname	string	Chat Entry Form Data: 'firstname'.
lastname	string	Chat Entry Form Data: 'lastname'.
email	string	Chat Entry Form Data: 'email'.
subject	string	Chat Entry Form Data: 'subject'.
userData	object	Arbitrary data to attach to the chat session (AKA attachedData). Properties defined here will be merged with default userData set in the configuration object.

Resolutions

Status	When	Returns
resolved	Server confirms session started	(AJAX Response Object)
rejected	A chat session is already active	'There is already an active chat session'
rejected	AJAX exception occurs	(AJAX Response Object)
rejected	Server exception occurs	(AJAX Response Object)
rejected	userData is invalid	'malformed data object provided in userData property'

endChat

Ends the chat session with the chat server via GES or with the service configured under transport section.

Example

```
oMyPlugin.command('WebChatService.endChat').done(function(e){  
    // WebChatService ended a chat successfully  
}).fail(function(e){  
    // WebChatService failed to end chat  
});
```

Resolutions

Status	When	Returns
resolved	Active session is ended successfully	(AJAX Response Object)

Status	When	Returns
rejected	No chat session is currently active	'There is no active chat session'

sendMessage

Sends a message from the client to the chat session.

Example

```
oMyPlugin.command('WebChatService.sendMessage', {message: 'hi'}).done(function(e){  
    // WebChatService sent a message successfully  
}).fail(function(e){  
    // WebChatService failed to send a message  
});
```

Options

Option	Type	Description
message	string	The message you want to send

Resolutions

Status	When	Returns
resolved	Message is successfully sent	(AJAX Response Object)
rejected	No message text provided	'No message text provided'
rejected	No chat session is currently active	'There is no active chat session'
rejected	AJAX exception occurs	(AJAX Response Object)

sendCustomNotice

Sends a custom notice from the client to the chat server. This request is used to deliver any custom notification between a custom client application and a custom agent desktop. Neither Genesys Widgets, nor Workspace, uses this out of the box.

Example

```
oMyPlugin.command('WebChatService.sendCustomNotice', {message: 'bye'}).done(function(e){  
    // WebChatService sent a custom message successfully  
}).fail(function(e){  
    // WebChatService failed to send a custom message  
});
```


Options

Option	Type	Description
message	string	A message you want to send along with the custom notice

Resolutions

Status	When	Returns	Introduced/Updated
resolved	Message is successfully sent	(AJAX Response Object)	
rejected	AJAX exception occurs	(AJAX Response Object)	
rejected	The server doesn't support receiving custom notices	This transport doesn't support sendCustomNotice command.	9.0.008.04

sendTyping

Sends a "*Customer typing*" notification to the chat session. A visual indication will be shown to the agent.

Example

```
oMyPlugin.command('WebChatService.sendTyping').done(function(e){  
    // WebChatService sent typing successfully  
}).fail(function(e){  
    // WebChatService failed to send typing  
});
```

Options

Option	Type	Description
Message	String	The message you want to send along with the typing notification

Resolutions

Status	When	Returns
resolved	AJAX request is successful	(AJAX Response Object)
rejected	AJAX exception occurs	(AJAX Response Object)
rejected	No chat session is currently active	'There is no active chat session'

sendFilteredMessage

Sends a message along with a regular expression to match the message and hide it from the client. Useful for sending codes and tokens through the WebChat interface to the Agent Desktop.

Important

Filters are now automatically stored and recalled on chat restore for the duration of the session.

Example

```
oMyPlugin.command('WebChatService.sendFilteredMessage', {
  message: 'filtered message',
  regex: /[a-zA-Z]/
}).done(function(e){
  // WebChatService sent filtered message successfully
}).fail(function(e){
  // WebChatService failed to send filtered message
});
```

Options

Option	Type	Description
message	string	Message you want to send but don't want to appear in the transcript
regex	RegExp	Regular expression to match the message

Resolutions

Status	When	Returns
resolved	There is an active session	n/a
rejected	No chat session is currently active	'No active chat session'

addPrefilter

Adds a new pre-filter regular expression to the pre-filter list. Any messages matched using the pre-filters will not be shown in the transcript

Important

Filters are now automatically stored and recalled on chat restore for the duration of the session.

Example

```
oMyPlugin.command('WebChatService.addPrefilter', {filters: /[a-zA-Z]/}).done(function(e){  
    // WebChatService added filter successfully  
    // e == Object of registered prefilters  
}).fail(function(e){  
    // WebChatService failed to add filter  
});
```

Options

Option	Type	Description
filters	RegExp or Array of RegExp	Regular Expression(s) to add to the prefilter list

Resolutions

Status	When	Returns
resolved	Valid filters are provided	Array of all registered prefilters.
rejected	Invalid or missing filters provided	'Missing or invalid filters provided. Please provide a regular expression or an array of regular expressions.'

updateUserData

Updates the userData properties associated with the chat session. If this command is called before a chat session starts, it will update the internal userData object and will be sent when a chat session starts. If this command is called after a chat session starts, a request to the server will be made to update the userData on the server associated with the chat session.

Example

```
oMyPlugin.command('WebChatService.updateUserData', {firstname: 'Joe'}).done(function(e){  
    // WebChatService updated user data successfully  
}).fail(function(e){  
    // WebChatService failed to update user data  
});
```

Options

Option	Type	Description
n/a	object	userData object you want to send to the server for this active session

Resolutions

Status	When	Returns	Introduced/Updated
resolved	Session is active and userData is successfully sent	(AJAX Response Object)	
rejected	Session is active and AJAX exception occurs	(AJAX Response Object)	
resolved	Session is not active and internal userData object is merged with new userData properties provided	The internal userData object that will be sent to the server	
rejected	Session is active and the server doesn't support updating userData	This transport doesn't support updating userData during an active chat session.	9.0.008.04

poll

Internal use only. Starts polling for new messages.

Example

```
oMyPlugin.command('WebChatService.poll').done(function(e){
    // WebChatService started polling successfully
}).fail(function(e){
    // WebChatService failed to start polling
});
```

Resolutions

Status	When	Returns	Introduced/Updated
resolved	There is an active session	n/a	
rejected	WebChatService isn't calling this command	'Access Denied to private command. Only WebChatService is allowed to invoke this command.'	
rejected	No chat session is	'previous poll has not	

Status	When	Returns	Introduced/Updated
	currently active	finished.'	
rejected	The server doesn't support polling	'This transport doesn't support polling.'	9.0.008.04

startPoll

Starts automatic polling for new messages.

Example

```
oMyPlugin.command('WebChatService.startPoll').done(function(e){  
    // WebChatService started polling successfully  
}).fail(function(e){  
    // WebChatService failed to start polling  
});
```

Resolutions

Status	When	Returns	Introduced / Updated
resolved	There is an active session	n/a	
rejected	No chat session is currently active	No active chat session	
rejected	The server doesn't support polling	This transport doesn't support polling	9.0.008.04

stopPoll

Stops automatic polling for new messages.

Example

```
oMyPlugin.command('WebChatService.stopPoll').done(function(e){  
    // WebChatService stopped polling successfully  
}).fail(function(e){  
    // WebChatService failed to stop polling  
});
```

Resolutions

Status	When	Returns	Introduced / Updated
resolved	There is an active session	n/a	
rejected	No chat session is	No active chat session	

Status	When	Returns	Introduced / Updated
	currently active		
rejected	The server doesn't support polling	This transport doesn't support polling	9.0.008.04

resetPollExceptions

Resets the poll exception count to 0. pollExceptionLimit is set in the configuration.

Example

```
oMyPlugin.command('WebChatService.resetPollExceptions').done(function(e){
    // WebChatService reset polling successfully
}).fail(function(e){
    // WebChatService failed to reset polling
});
```

Resolutions

Status	When	Returns	Introduced / Updated
resolved	Always	n/a	
rejected	The server doesn't support polling	This transport doesn't support resetPollExceptions command.	9.0.008.04

restore

Internal use only. You should not invoke this manually unless you are using Async mode.

Example

```
oMyPlugin.command('WebChatService.restore').done(function(e){
    // WebChatService restored successfully
}).fail(function(e){
    // WebChatService failed to restore
});
```

Options

Option	Type	Description	Accepted Values	Introduced / Updated
sessionData	string	The session data that is needed to restore the WebChat in Async	(JWT string token)	9.0.008.04

Option	Type	Description	Accepted Values	Introduced / Updated
		mode. It is a JWT token string value. Applicable only when using WebChat with Genesys Multicloud CX v3 API. For more information, see the “Genesys Multicloud CX v3” tab in the “Options” table in configuration options.		

Resolutions

Status	When	Returns	Introduced / Updated
resolved	Session has been found	n/a	
rejected	Session cannot be found	n/a	
rejected	Restoring chat session is in progress	Already restoring. Ignoring request.	9.0.002.06
rejected	Chat session is already active	Chat session is already active, ignoring restore command.	9.0.002.06
rejected	Trying restore chat session manually	Access Denied to private command. Only WebChatService is allowed to invoke this command in Non-Async mode.	9.0.002.06

getTranscript

Fetches an array of all messages in the chat session.

Important

For more information on the fields included in JSON response, see Digital Channels Chat V2 Response Format.

Example

```
oMyPlugin.command('WebChatService.getTranscript').done(function(e){  
    // WebChatService got transcript successfully
```

```
        // e == Object with an array of messages
    }).fail(function(e){
        // WebChatService failed to get transcript
    });
```

Resolutions

Status	When	Returns
resolved	Always	Object with an array of messages

getAgents

Return a list of agents currently participating in the chat. Includes agent metadata.

Example

```
oMyPlugin.command('WebChatService.getAgents').done(function(e){
    // WebChatService got agents successfully
    // e == Object with agents information in chat
}).fail(function(e){
    // WebChatService failed to get agents
});
```

Resolutions

Status	When	Returns
resolved	Always	(Object List) {name: (String), connected: (Boolean), supervisor: (Boolean), connectedTime: (int time), disconnectedTime: (int time)}

getStats

Returns stats on chat session including start time, end time, duration, and list of agents.

Example

```
oMyPlugin.command('WebChatService.getStats').done(function(e){
    // WebChatService got stats successfully
    // e == Object with chat session stats
}).fail(function(e){
    // WebChatService failed to get stats
});
```


Resolutions

Status	When	Returns
resolved	Always	{agents: (Object), startTime: (int time), endTime: (int time), duration: (int time)}

sendFile

[Introduced: 9.0.008.04]

Sends the file from the client machine to the agent.

Example

```
oMyPlugin.command('WebChatService.sendFile', {files: $('').attr('type', 'file') /* Only works on UI, can not dynamically change */ }).done(function(e){  
    // WebChatService sent file successfully  
}).fail(function(e){  
    // WebChatService failed to send file  
});
```

Options

Option	Type	Description
files	File	A reference to a file input element (for example)

Resolutions

Status	When	Returns
resolved	The file sent is a valid type and size	(AJAX Response Object)
rejected	The file sent is an invalid type	(AJAX Response Object)
rejected	The number of uploads is exceeded	(AJAX Response Object)
rejected	The file size exceeds the limit	(AJAX Response Object)
rejected	The file size is too large or an unknown error occurs	(AJAX Response Object)
rejected	The server doesn't support file uploads	This transport doesn't support file uploads

downloadFile

Downloads the file to the client machine. Example

```
oMyPlugin.command('WebChatService.downloadFile', {fileId: '1', fileName:
'myfile.txt'}).done(function(e){

    // WebChatService sent file successfully

}).fail(function(e){

    // WebChatService failed to send file

});
```

Options

Option	Type	Description
field	string	This is the id of the file to be downloaded from the session

Resolutions

Status	When	Returns
resolved	The file is downloaded successfully	n/a

getSessionData

[Introduced: 9.0.002.06]

Retrieves the active session data at any time.

Example

```
oMyPlugin.command('WebChatService.getSessionData')
```

Resolutions

Status	When	Returns	Introduced / Updated
resolved	Always, when using Chat via GMS API. For more information, see the 'GMS' tab in the 'Options' table in configuration options.	{secureKey: (string), sessionId: (number/string), alias: (number/string), userId: (number/string)}	
resolved	Always, when using Chat via Genesys Multicloud CX v3 API. For more information, see the 'Genesys Multicloud CX v3' tab in the 'Options' table in	{participantId: (string), sessionId: {string}, token: (string), transportId: (string)}	9.0.008.04

Status	When	Returns	Introduced / Updated
	configuration options.		
rejected	Never	undefined	

fetchHistory

[Introduced: 9.0.008.04]

This applies only in Asynchronous mode to fetch older chat messages. It does not fetch all of the messages at once; rather a certain number of messages are fetched every time this command is called. Response data will be available in the messageReceived event. This internal command determines the last received message index and, based on this information, fetches older messages whenever it is called.

Example

```
oMyPlugin.command('WebChatService.fetchHistory')
```

Resolutions

Status	When	Returns
resolved	Old messages are retrieved	(AJAX Response Object)
rejected	Request fails	(AJAX Response Object)
rejected	Asynchronous mode is not enabled	Fetching history messages applies only to Asynchronous chat
rejected	All messages are received	No more messages to fetch

registerTypingPreviewInput

Selects an HTML input to watch for key events. Used to trigger startTyping and stopTyping automatically.

Example

```
oMyPlugin.command('WebChatService.registerTypingPreviewInput', {input: $('input')
}).done(function(e){
    // WebChatService registered input area successfully
}).fail(function(e){
    // WebChatService failed to register typing preview
});
```

Options

Option	Type	Description
input	HTML Reference	An HTML reference to a text or textarea input

Resolutions

Status	When	Returns
resolved	Valid HTML input reference is provided	n/a
rejected	Invalid or missing HTML input reference	'Invalid value provided for the 'input' property. An HTML element reference to a textarea or text input is required.'

registerPreProcessor

Registers a function that receives the message object, allowing you to manipulate the values before it is rendered in the transcript.

Example

```
oMyPlugin.command('WebChatService.registerPreProcessor', {preprocessor: function(message){
    message.text = message.text + ' some preprocessing text';
    return message;
} }).done(function(e){
    // WebChatService registered preprocessor function
    // e == function that was registered
}).fail(function(e){
    // WebChatService failed to register function
});
```

Options

Option	Type	Description
preprocessor	function	The preprocessor function you want to register.

Resolutions

Status	When	Returns
resolved	A valid preprocessor function is provided and is registered	The registered preprocessor function.
rejected	An invalid preprocessor function is provided	No preprocessor function provided. Type provided was ''.

verifySession

Checks for existing WebChat session before triggering a proactive invite.

```
oMyPlugin.command('WebChatService.verifySession').done(function(e){
    if(e.sessionActive) {
        // dont show chat invite
    } else if(!e.sessionActive) {
        if(oMyPlugin.data('WebChat.open') == false){
```

```
        } else { // show chat invite
        } // dont trigger chat invite
    }
}).fail(function(e){
    // verifySession not supported for the transport
});
```

Resolutions

Status	When	Returns
resolved	A session exists or not	A boolean <i>sessionActive</i> , which holds the session state
rejected	The <i>verifySession</i> command is not supported for this transport	This transport doesn't support the <i>verifySession</i> command

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');
oMyPlugin.subscribe('WebChatService.ready', function(e){});
```

Name	Description	Data	Introduced/updated
ready	WebChatService is initialized and ready to accept commands.	n/a	
restored	Chat session has been restored after page navigation or refresh. In Asynchronous mode, this event includes data indicating whether a chat session has been restored in Async mode or not.	{async: (boolean)}	9.0.002.06
restoreTimeout	Chat session restoration attempted was denied after user navigated	n/a	

Name	Description	Data	Introduced/updated
	away from originating website for longer than the time limit: default 60 seconds.		
restoreFailed	Could not restore chat session after page navigation or refresh.	n/a	
restoredOffline	Chat session was restored normally but chat server is offline. This means no messages can come through. When chat server is comes back online, 'chatServerBackOnline' is published.	n/a	
messageReceived	A new message has been received from the server. Includes text messages, status messages, notices, and other message types.	{originalMessages: (object), messages: (array of objects), restoring: (boolean), sessionData: (object)}	9.0.002.06
error	An error occurred between the client and the server.	(AJAX Response)	
started	Chat session has successfully started.	(AJAX Response containing session data)	
ended	Chat session has successfully ended.	n/a	
agentTypingStarted	Agents has started typing a new message.	(AJAX Response)	
agentTypingStopped	Agent has stopped typing.	(AJAX Response)	
pollingStarted	Chat server automatic polling has started.	n/a	
pollingStopped	Chat server automatic polling has stopped.	n/a	
clientConnected	Indicates the user has been connected to the chat session.	{message: (object), agents: (object), numAgentsConnected: (number)}	
clientDisconnected	Indicates the user has been disconnected form the chat session.	{message: (object), agents: (object), numAgentsConnected: (number)}	
agentConnected	Indicates an agent has connected to the chat.	{message: (object), agents: (object), numAgentsConnected:	

Name	Description	Data	Introduced/updated
		(number)}	
agentDisconnected	Indicates an agent has disconnected from the chat.	{message: (object), agents: (object), numAgentsConnected: (number)}	
supervisorConnected	Indicates a supervisor has connected to the chat.	{message: (object), agents: (object), numAgentsConnected: (number)}	
supervisorDisconnected	Indicates a supervisor has disconnected from the chat.	{message: (object), agents: (object), numAgentsConnected: (number)}	
botConnected	Indicates a bot has connected to the chat. Important This event is applicable only when using v2 API.	{message: (object), agents: (object), numAgentsConnected: (number)}	9.0.014.13
botDisconnected	Indicates a bot has disconnected from the chat. Important This event is applicable only when using v2 API.	{message: (object), agents: (object), numAgentsConnected: (number)}	9.0.014.13
clientTypingStarted	The user has started typing. Sends an event to the agent.	n/a	
clientTypingStopped	After a user stops typing, a countdown begins. When the countdown completes, the typing notification will clear for the agent.	n/a	
disconnected	Cannot reach servers. No connection. Either the user is offline or the server is offline.	n/a	
reconnected	Connection restored. This event is only published after 'disconnected'.	n/a	
chatServerWentOffline	Chat server has gone offline but chat session has not ended. New messages are temporarily unavailable. This event is published only after the	n/a	

Name	Description	Data	Introduced/updated
	<p>configuration option 'pollExceptionLimit' has been exceeded. Default limit is 5 poll exceptions.</p> <p>'restoredOffline' is an alternate to this event that is used only when the chat server is down while trying to restore your chat session. The reason for having two events is to allow for separate handling of both scenarios.</p> <p>Important This event is applicable only when using v2 API.</p>		
chatServerBackOnline	<p>Chat server had come back online after going offline. This will only be published after 'chatServerWentOffline'.</p> <p>Important This event is applicable only when using v2 API.</p>	n/a	
connectionPending	<p>If there is a connection problem and WebChatService is trying to reconnect, this event will be published. Published before 'chatServerWentOffline'.</p> <p>Important This event is applicable only when using v2 API.</p>	n/a	
connectionRestored	<p>Is published when the connection has been reestablished. Publishes at the same time as 'chatServerBackOnline'.</p>	n/a	

Calendar

Contents

- **1 Overview**
 - 1.1 Usage
 - 1.2 Customization
 - 1.3 Namespace
 - 1.4 Mobile support
 - 1.5 Screenshots
- **2 Configuration**
 - 2.1 Description
 - 2.2 Example
 - 2.3 Options
- **3 Localization**
 - 3.1 Usage
 - 3.2 Example i18n JSON
- **4 API commands**
 - 4.1 configure
 - 4.2 generate
 - 4.3 showAvailability
 - 4.4 reset
- **5 API events**

- Developer

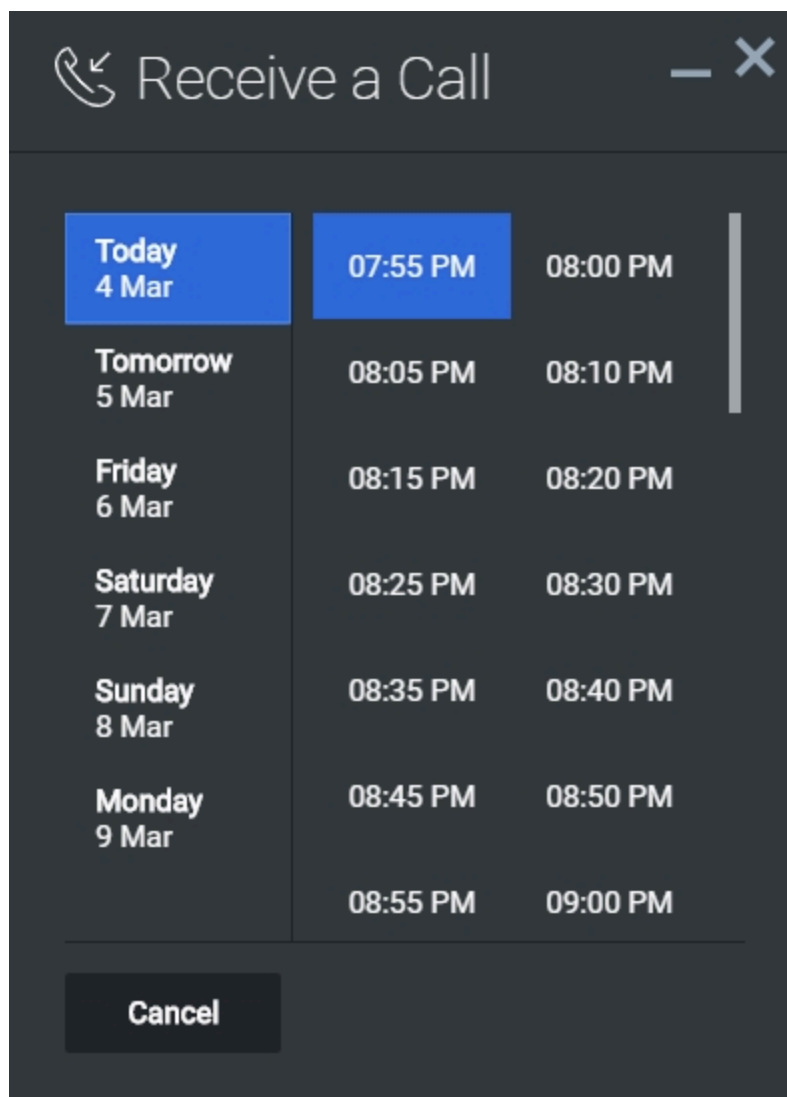
Learn how to display a calendar, so your customers can choose when they want to be contacted.

Related documentation:

-

Overview

The Calendar UI plugin displays time slots for a selected day. The number of days to display—and the opening and closing times for a day—are configurable, as shown in the configuration section.



Usage

Important

By default, the Calendar widget needs a UI container to display itself properly. For information about how to create and display a calendar, see the API events section.

- Enable or disable certain sections of a day using `calendarHours.section.enable`
- Define your own business hours for each section of a day using `calendarHours.section.openTime` and `calendarHours.section.closeTime`
- Use `showAvailability` to enable only those time slots for which a customer service agent is available and

disable the rest.

- Define your own time interval between each time slot.

How does the Calendar widget render time slots in local time zones?

1. The Calendar widget uses the command `showAvailability` which calls `CallbackService.availability` with the start date. This start date is then converted into the ISO 8601 format, using UTC as the timezone by `toISOString()`, internally.
2. The Callback service fetches the available time slots from the server.
3. The Calendar gets the available time slots from `CallbackService.availableSlots` in the ISO 8601 format, using UTC as the timezone.
4. Each and Every Time Slot is converted according to the user's local time zone internally through `Date()` and `toString()` methods in the Calendar Plugin.

Customization

All the texts displayed by the Calendar Widget are fully localizable.

Namespace

The Calendar plugin has the following namespaces tied to each of the following types:

Type	Namespace
Configuration	calendar
i18n - Localization	calendar
CXBus—API commands & API events	Calendar
CSS	.cx-calendar

Mobile support

Calendar supports both desktop and mobile devices. Like all Genesys Widgets, there are two main modes: Desktop and Mobile. Desktop is used for monitors, portable computers, and tablets, while Mobile is used for mobile devices. When a mobile device is detected, Calendar switches to special full-screen templates that are optimized for both portrait and landscape orientations.

Switching between Desktop and Mobile mode is done automatically by default. You can also configure Genesys Widgets to switch between Desktop and Mobile mode manually.

Screenshots

Dark theme

Calendar



Light theme



Configuration

Description

Calendar shares the **_genesys.widgets.calendar** configuration namespace. It also has UI options.

Example

```
window._genesys.widgets.calendar = {  
  showAvailability: true,  
  numberOfDays: 5,  
  hideUnavailableTimeSlots: false  
  
  calendarHours: {  
    interval: 10,  
    allDay: {  
      openTime: '09:00',  
      closeTime: '23:59'  
    }  
  }  
};
```

Options

Name	Type	Description	Default	Required
showAvailability	boolean	Enable or disable calendar to update the time slots based on the callback availability. The unavailable time slots are grayed out.	true	n/a
numberOfDays	number	The number of days to display on calendar starting today.	5	n/a
timeFormat	number/string	This sets the time format for the timestamps in this widget. It can be 12 or 24.	12	n/a
hideUnavailableTimeSlots	boolean	Show hide the unavailable callback time slots.	false	n/a
calendarHours.interval	number	The time interval between each consecutive time slot displayed on calendar.	15	n/a
calendarHours.allDay.openTime	number	Opening time in 'HH:MM' 24 Hr format.	17:00	n/a
calendarHours.allDay.closeTime	number	Closing time in 'HH:MM' 24 Hr format.	23:59	n/a

Localization

Important

For information on how to set up localization, refer to [Localize widgets and services](#).

Usage

You must use the **calendar** namespace when you're defining localization strings for the Calendar plugin in your i18n JSON file.

The following example shows how to define new strings for the **en** (English) language. You can use any language codes you wish, as there isn't a standard format. When selecting the active language in your configuration, you must match one of the language codes defined in your i18n JSON file. Note that you must only define a language code once in your i18n JSON file. Inside each language object you must define new strings for each widget.

Example i18n JSON

```
{
  "en": {
    "calendar": {
      "CalendarDayLabels": [
        "Sunday",
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday"
      ],
      "CalendarMonthLabels": [
        "Jan",
        "Feb",
        "Mar",
        "Apr",
        "May",
        "Jun",
        "Jul",
        "Aug",
        "Sept",
        "Oct",
        "Nov",
        "Dec"
      ],
      "CalendarLabelToday": "Today",
      "CalendarLabelTomorrow": "Tomorrow",
      "CalendarTitle": "Schedule a Call",
      "CalendarOkButtonText": "Okay",
      "CalendarError": "Unable to fetch availability details.",
      "CalendarClose": "Cancel",
      "AriaWindowTitle": "Calendar Window",
      "AriaCalendarClose": "Cancel",
      "AriaYouHaveChosen": "You have chosen",
      "AriaNoTimeSlotsFound": "No time slots found for selected date"
    }
  }
}
```

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, you must not use the global bus object to register your custom plugins. For more information about extending Genesys Widgets, see Genesys Widgets Extensions.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('Calendar.reset');
```

configure

Internal use only. The main App plugin shares widget configuration settings using each widget's **configure** command. The **configure** command can only be called once, at startup. If you call **configure** after startup, the results are unpredictable.

generate

Builds and generates the calendar. Subscribe to the **generate** events to get the generated calendar and display it where you would like to.

Example

```
oMyPlugin.command('Calendar.generate', {date: 'Mon Mar 20 2017 19:51:47 GMT-0700  
(PDT)'}).done(function(e){  
    // Calendar generated successfully  
}).fail(function(e){  
    // Calendar failed to generate  
});
```

Options

Option	Type	Description
date	Date string/object	To pre-select the date and time on calendar.

Resolutions

Status	When	Returns
resolved	When the calendar is successfully generated	n/a
rejected	When Invalid date is passed to calendar	'Invalid data'

showAvailability

Update the calendar time slots with the callback availability. This enables only those time slots that have the callback facility and disables the rest.

Example

```
oMyPlugin.command('Calendar.showAvailability', {date: '03/22/17'}).done(function(e){  
    // Calendar showed availability successfully  
}).fail(function(e){  
    // Calendar failed to show availability  
});
```

Options

Option	Type	Description
date	Date string/object	Update the available time slots in the Calendar plugin for the selected Date. Note that, after calling this command, the internal showAvailability value is set to true for this session and the Calendar only shows the available time slots when switching between other dates.

Resolutions

Status	When	Returns
resolved	When time slots are successfully updated	n/a
rejected	When no date value is found to	'No date found to check'

Status	When	Returns
	check the availability	availability'
rejected	When invalid date value is found	'Invalid date'

reset

Resets the calendar with no pre-selected values.

Example

```
oMyPlugin.command('Calendar.reset').done(function(e){
    // Calendar reset successfully
}).fail(function(e){
    // Calendar failed to reset
});
```

Resolutions

Status	When	Returns
resolved	When calendar is successfully reset	n/a

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, you must not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');
```

Calendar

```
oMyPlugin.subscribe('Calendar.ready', function(e){});
```

Name	Description	Data
ready	Calendar is initialized and ready to accept commands.	n/a
generated	Calendar UI has been generated. Use this event to get the calendar UI and display where you would like to.	{ ndCalendar: } <div></div>
selectedDateTime	Date and time selected on calendar.	{ dayString: , dateString: , timeString: , date: }

Callback

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Dependency](#)
 - [1.3 Customization](#)
 - [1.4 Namespace](#)
 - [1.5 Mobile support](#)
 - [1.6 Screenshots](#)
- [2 Configuration](#)
 - [2.1 Example](#)
 - [2.2 Options](#)
- [3 Localization](#)
 - [3.1 Usage](#)
 - [3.2 Example i18n JSON](#)
- [4 API commands](#)
 - [4.1 open](#)
 - [4.2 close](#)
 - [4.3 minimize](#)
 - [4.4 showOverlay](#)
 - [4.5 hideOverlay](#)
 - [4.6 configure](#)
- [5 API events](#)
- [6 Metadata](#)
 - [6.1 Interaction Lifecycle](#)
 - [6.2 Lifecycle scenarios](#)
 - [6.3 Metadata](#)
- [7 Customizable Callback registration form](#)
 - [7.1 Default example](#)
 - [7.2 Properties](#)

- [7.3 Labels](#)
- [7.4 Wrappers](#)
- [7.5 Validation](#)
- [7.6 Form submit](#)
- [7.7 Form pre-fill](#)

- Developer

Learn how to use the Callback Widget to fetch user details.

Related documentation:

-

[Link to video](#)

Overview

Important

This documentation relies on Genesys Callback APIs available to Engage Cloud customers. The only supported version is v3 as exposed by Engagement API.

The Callback Widget provides a form to fetch user details such as name, phone number, and email—and whether the customer would like an immediate callback or would prefer to receive a call at another time of their choosing. Callback then submits this information to Customer Service. The times that Callback displays are based on agent availability, meaning the user can select a time that works for everyone.

Usage

Use the following methods to launch Callback manually:

- Call the **Callback.open** command
- Configure ChannelSelector so that *Receive a Call* appears as a channel
- Configure Calendar to show a Date-Time picker for selecting a preferred time

Dependency

The Callback Widget requires the Calendar plugin.

Customization

You can customize and localize all of the text shown in the Callback Widget by adding entries into your configuration and localization options.

Callback supports themes. You can create and register your own themes for Genesys Widgets.

Namespace

The Callback plugin has the following namespaces tied up with each of the following types:

Type	Namespace
Configuration	callback
i18n—Localization	callback
CXBus— API commands & API events	Callback
CSS	.cx-callback

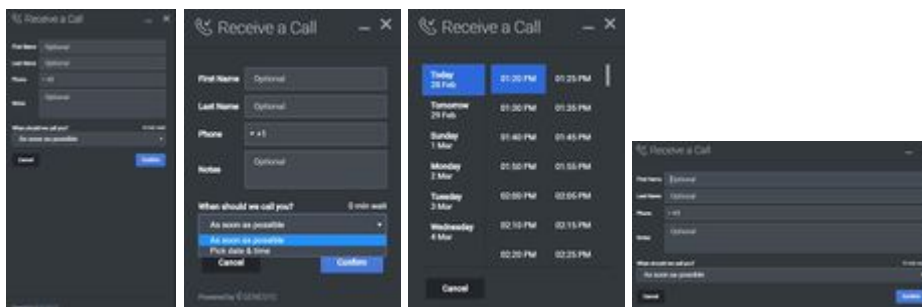
Mobile support

Callback supports both desktop and mobile devices. Like all Genesys Widgets, there are two main modes: Desktop & Mobile. Desktop is employed for monitors, laptops, and tablets. Mobile is employed for smartphones. When a smartphone is detected, Callback switches to special full-screen templates that are optimized for both portrait and landscape orientations.

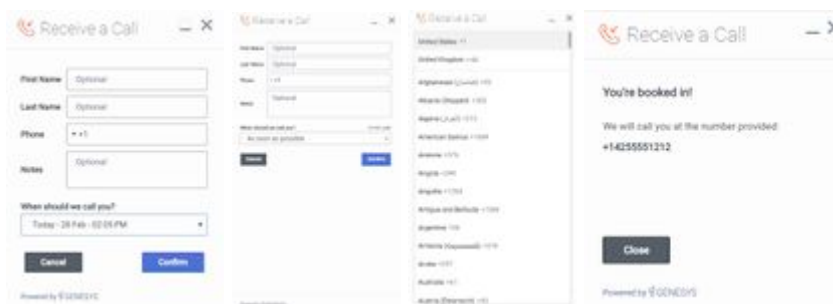
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

Dark theme



Light theme



Configuration

Callback and CallbackService share the **_genesys.widgets.callback** configuration namespace. Callback has UI options while CallbackService has connection options.

Example

```
window._genesys.widgets.callback = {  
  apikey: 'n3eNkgXXXXXXXXXXXXXXXXXA',  
  dataURL: 'http://host:port/genesys/1/service/callback/samples',  
  userData: {},  
  countryCodes: true,  
  immediateCallback: true,  
  scheduledCallback: true,  
  ewt: {  
    display: true,  
    queue: 'chat_ewt_test',  
    threshold: 2000,  
    immediateCallback: {  
      thresholdMin: 1000,  
      thresholdMax: 3000  
    }  
  }  
};
```

Options

Name	Type	Description	Default	Required
countryCodes	boolean	Enable/disable display of country codes for phone number.	true	n/a
immediateCallback	boolean	Enable/disable the immediate (As Soon As Possible) callback option.	true	n/a
scheduledCallback	boolean	Enable/disable the scheduling (Pick date & time) callback option.	true	n/a
form	object	An object containing a custom registration form definition. The definition placed here becomes the default registration form layout for Callback. See Customizable Callback	A basic registration form is defined internally by default	n/a

Name	Type	Description	Default	Required
		Registration Form.		
ewt.display	boolean	To display Estimated Wait Time (EWT) details.	true	n/a
ewt.queue	string	EWT service channel virtual queue.	none	Always required if Estimated Waiting Time has to be displayed.
ewt.threshold	number	If EWT is less than this threshold value (seconds), wait time will not be shown.	30	n/a
ewt.refreshInterval	number	EWT is updated for every time interval (seconds) defined here.	10	n/a
ewt.immediateCallbackThresholdMin	number	If EWT is less than this minimum threshold value (seconds), then 'As Soon As Possible' option (Immediate Callback) will be disabled. This value should be configured less than or equal to above ewt.threshold value.	none	n/a
ewt.immediateCallbackThresholdMax	number	If EWT is more than this maximum threshold value (seconds), then 'As Soon As Possible' option (Immediate Callback) will be disabled.	none	n/a

Localization

Important

For information on how to set up localization, please refer to [Localize widgets and](#)

services.

Usage

Use the **callback** namespace when defining localization strings for the Callback plugin in your i18n JSON file.

The following example shows how to define new strings for the **en** (English) language. You can use any language codes you wish; there is no standard format. When selecting the active language in your configuration, you must match one of the language codes defined in your i18n JSON file. Please note that you must only define a language code once in your i18n JSON file. Inside each language object you should define new strings for each widget.

Example i18n JSON

```
{
  "en": {
    "callback": {
      "CallbackTitle": "Receive a Call",
      "CancelButtonText": "Cancel",
      "AriaCancelButtonText": "Cancel",
      "ConfirmButtonText": "Confirm",
      "AriaConfirmButtonText": "Confirm",
      "CallbackPlaceholderRequired": "Required",
      "CallbackPlaceholderOptional": "Optional",
      "CallbackFirstName": "First Name",
      "CallbackLastName": "Last Name",
      "CallbackPhoneNumber": "Phone",
      "CallbackQuestion": "When should we call you?",
      "CallbackDayLabels": [
        "Sunday",
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday"
      ],
      "CallbackMonthLabels": [
        "Jan",
        "Feb",
        "Mar",
        "Apr",
        "May",
        "Jun",
        "Jul",
        "Aug",
        "Sep",
        "Oct",
        "Nov",
        "Dec"
      ],
      "CallbackConfirmDescription": "You're booked in!",
      "CallbackNumberDescription": "We will call you at the number",
      "CallbackNotes": "Notes",
      "provided": ""
    }
  }
}
```

```

    "CallbackDone": "Close",
    "AriaCallbackDone": "Close",
    "CallbackOk": "Okay",
    "AriaCallbackOk": "Okay",
    "CallbackCloseConfirm": "Are you sure you want to cancel arranging
this callback?",
    "CallbackNoButtonText": "No",
    "AriaCallbackNoButtonText": "No",
    "CallbackYesButtonText": "Yes",
    "AriaCallbackYesButtonText": "Yes",
    "CallbackWaitTime": "Wait Time",
    "CallbackWaitTimeText": "min wait",
    "CallbackOptionASAP": "As soon as possible",
    "CallbackOptionPickDateTime": "Pick date & time",
    "AriaCallbackOptionPickDateTime": "Opens a date picker",
    "CallbackPlaceholderCalendar": "Select Date & Time",
    "AriaMinimize": "Callback Minimize",
    "AriaWindowLabel": "Callback Window",
    "AriaMaximize": "Callback Maximize",
    "AriaClose": "Callback Close",
    "AriaCalendarClosedStatus": "Calendar is closed",
    "Errors": {
      "501": "Invalid parameters cannot be accepted, please check
the supporting server API documentation for valid parameters.",
      "503": "Missing apikey, please ensure it is configured
properly.",
      "1103": "Missing apikey, please ensure it is configured
properly.",
      "7030": "Please enter a valid phone number.",
      "7036": "Callback to this number is not possible. Please
retry with another phone number.",
      "7037": "Callback to this number is not allowed. Please retry
with another phone number.",
      "7040": "Please configure a valid service name.",
      "7041": "Too many requests at this time.",
      "7042": "Office closed. Please try scheduling within the
office hours.",
      "unknownError": "Something went wrong, we apologize for the
inconvenience. Please check your connection settings and try again.",
      "phoneNumberRequired": "Phone number is required."
    }
  }
}

```

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

Callback

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('Callback.open');
```

open

Opens the Callback UI.

Example

```
oMyPlugin.command('Callback.open', {  
    form: {  
        autoSubmit: false,  
        firstname: 'John',  
        lastname: 'Smith',  
        subject: 'Customer Satisfaction',  
        desiredTime: 'now',  
        phonenumber: '8881110000'  
    },  
    formJSON: {...}  
}).done(function(e){  
    // Callback opened successfully  
}).fail(function(e){  
    // Callback failed to open  
});
```

Options

Option	Type	Description
form	object	Object containing form data to prefill in the callback form and optionally auto-submit the form.
form.autoSubmit	boolean	Automatically submit the callback form.
form.firstname	string	Value for the first name entry field.
form.lastname	string	Value for the last name entry field.
form.subject	string	Value for the notes entry field.
form.desiredTime	string	This value is shared by the immediate or scheduled callback drop down option in the form (in other words, As Soon As Possible or Pick date & time). A string value 'now' pre-selects the 'As Soon As Possible' option. A string value with Date Time or Date Object, is passed into this drop down option and pre-selected.

Option	Type	Description
		During form submission, it is converted into UTC string format and sent to the server as the desired callback time.
form.phonenumber	string	Value for the phone entry field. Should be a valid telephone number, when used with a prefix '+' auto selects the country flag near the phone input field.
formJSON	object	An object containing a custom registration form definition. See Customizable Callback Registration Form.
userData	object	Arbitrary data that is to be attached with callback schedule. Properties defined here will be merged with default userData set in the configuration object.

Resolutions

Status	When	Returns
resolved	Callback form is successfully opened	n/a
rejected	Callback form is already open	'already opened'

close

Closes the Callback UI.

Example

```
oMyPlugin.command('Callback.close');
```

Resolutions

Status	When	Returns
resolved	Callback form is successfully closed	n/a
rejected	Callback form is already closed	'already closed'
rejected	User has entered some details on the form and trying to close it without confirming cancellation	'User must confirm close'

minimize

Minimizes or un-minimizes the Callback UI.

Example

```
oMyPlugin.command('Callback.minimize');
```

Options

Option	Type	Description
minimized	boolean	Rather than toggling the current minimized state you can specify the minimized state directly: true = minimized, false = unminimized.

Resolutions

Status	When	Returns
resolved	Always	n/a
rejected	Never	n/a

showOverlay

Displays a slide-down overlay over the Callback's content. You can fill this overlay with disclaimers, articles and other information.

Example

```
oMyPlugin.command('Callback.showOverlay', {
    html: '
Example text
',
});
```

Options

Option	Type	Description
html	string or HTML reference	The HTML content you want to display in the overlay.
hideFooter	boolean	Normally the overlay appears

Option	Type	Description
		between the titlebar and footer bar. Set this to true to have the overlay overlap the footer to gain a bit more vertical space. This should only be used in special cases. For general use, don't set this value.

Resolutions

Status	When	Returns
resolved	Callback is open and the overlay opens	n/a
rejected	Callback is not currently open	Callback is not currently open. Ignoring command.

hideOverlay

Hides the slide-down overlay.

Example

```
oMyPlugin.command('Callback.hideOverlay');
```

Resolutions

Status	When	Returns
resolved	Callback is open and the overlay closes	n/a
rejected	Callback is not currently open	Callback is not currently open. Ignoring command.

configure

Internal use only. The main App plugin shares configuration settings to widgets using each widget's configure command. The configure command can only be called once at startup. Calling configure again after startup may result in unpredictable behavior.

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');
oMyPlugin.subscribe('Callback.ready', function(e){});
```

Name	Description	Data
opened	The Callback widget has appeared on screen.	Metadata
ready	Callback is initialized and ready to accept commands.	n/a
started	When the user has started filling out the Callback widget form or auto pre-filled it.	Metadata
submitted	When the user has submitted the form.	Metadata
completed	When the Callback widget form is submitted successfully.	Metadata
cancelled	When the user has abandoned the interaction by closing the Callback widget before scheduling a callback.	Metadata
closed	The Callback widget has been removed from the screen.	Metadata

Metadata

Interaction Lifecycle

Every Callback interaction has a sequence of events we describe as the *Interaction Lifecycle*. This is a sequence of events that tracks progress and choices from the beginning of an interaction (opening Callback), to the end (closing Callback), and every step in between.

The following events are part of the Interaction Lifecycle:

```
ready
opened
started
submitted
```

cancelled
completed
closed

Lifecycle scenarios

An Interaction Lifecycle can vary, based on each user's intent and experience with Callback. Here are several sequences of events in the lifecycle that correspond to different scenarios.

The user opened Callback but changed their mind and closed it without entering any information:

ready -> opened -> cancelled -> closed

The user started filling out the form but closed Callback without submitting the callback request:

ready -> opened -> started -> cancelled -> closed

The user started filling out the form and submitted it successfully:

ready -> opened -> started -> submitted -> completed -> closed

Tip

For a list of all Callback events, see API events.

Metadata

Each event in the Interaction Lifecycle includes the following block of metadata. By default, all values are set to false. As the user progresses through the lifecycle of a Callback interaction, these values will be updated.

The metadata block contains boolean state flags, counters, timestamps, and elapsed times. These values can be used to track and identify trends or issues with callback interactions. During run-time, the metadata can help you offer a smart and dynamic experience to your users.

Reference

Name	Type	Description
proactive	boolean	Indicates Callback was offered and accepted proactively.
prefilled	boolean	Indicates the form was prefilled with info automatically.
autoSubmitted	boolean	Indicates the form was submitted automatically, usually after being prefilled.
errors	array/boolean	An array of error codes encountered after submitting the form. If no errors, this value will be false.

Name	Type	Description
form	object	An object containing the form parameters when the form is submitted.
opened	integer (timestamp)	Timestamp indicating when Callback was opened.
started	integer (timestamp)	Timestamp indicating when the user started entering information into the form.
cancelled	integer (timestamp)	Timestamp indicating when the callback request is cancelled. Cancelled refers to when a user abandoned the interaction by closing Callback before scheduling a callback.
completed	integer (timestamp)	Timestamp indicating when the callback request was sent successfully.
closed	integer (timestamp)	Timestamp indicating when Callback was closed.
elapsed	integer (milliseconds)	Total elapsed time in milliseconds from when the user started entering information to when the user cancelled or completed the interaction.

Customizable Callback registration form

Callback allows you to customize the registration form shown to users prior to starting a session. The following form inputs are currently supported:

- Text
- Select
- Hidden
- Checkbox
- Textarea

Customization is done through an object definition that defines the layout, input type, label, and attributes for each input. You can set the default registration form definition in the `_genesys.widgets.callback.form` configuration option. Alternately, you can pass a new registration form definition through the `Callback.open` command:

```
_genesys.widgets.bus.command("Callback.open", {formJSON: oRegFormDef});
```

Inputs are rendered as stacked rows with one input and one optional label per row.

Default example

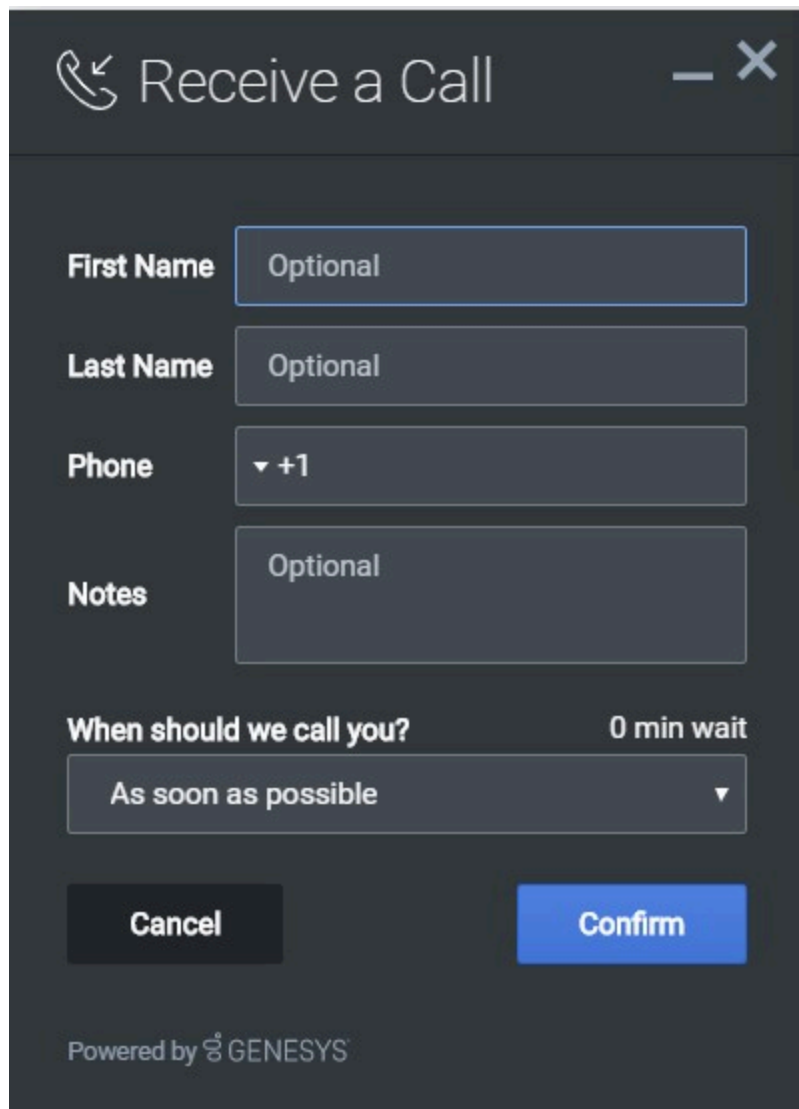
The following example is the default object used to render Callback's registration form. This is a very simple definition that does not use many properties.

Important

The Phone Number field with name **phonenumber** is required for all Callback custom forms. This field value is required by Genesys Callback API to schedule a Callback.

```
{
  wrapper: "
", inputs: [ { id: "cx_form_callback_firstname", name: "firstname", maxlength:
"100", placeholder: "@i18n:callback.CallbackPlaceholderOptional", label:
"@i18n:callback.CallbackFirstName" }, { id: "cx_form_callback_lastname", name:
"lastname", maxlength: "100", placeholder:
"@i18n:callback.CallbackPlaceholderOptional", label:
"@i18n:callback.CallbackLastName" }, { id: "cx_form_callback_phone_number",
name: "phonenumber", maxlength: "14", placeholder:
"@i18n:callback.CallbackPlaceholderRequired", label:
"@i18n:callback.CallbackPhoneNumber", onkeypress: function(event) { // To
allow only number inputs return (event.charCode >= 48 && event.charCode
```

Using this definition will result in this output:



The screenshot shows a dark-themed modal window titled "Receive a Call" with a close button (X) in the top right corner. The form contains the following fields:

- First Name**: A text input field with the placeholder text "Optional".
- Last Name**: A text input field with the placeholder text "Optional".
- Phone**: A text input field with a dropdown arrow and the placeholder text "+1".
- Notes**: A large text area with the placeholder text "Optional".
- When should we call you?**: A label above a dropdown menu.
- 0 min wait**: A label to the right of the dropdown menu.
- As soon as possible**: The selected option in the dropdown menu, with a dropdown arrow.

At the bottom of the form are two buttons: "Cancel" (dark grey) and "Confirm" (blue). Below the buttons, it says "Powered by GENESYS" with the Genesys logo.

Important

Form fields with id **cx_form_schedule_options** and **cx_form_schedule_time** are not customizable.

Properties

Each input definition can contain any number of properties. These are categorized in two groups: *Special Properties*, which are custom properties used internally to handle rendering logic, and *HTML Attributes* which are properties that are applied directly as HTML attributes on the input element.

Special properties

Property	Type	Default	Description
type	string	"text"	Sets the type of input to render. Possible values are currently text, hidden, select, checkbox, and textarea.
label	string		Set the text for the label. If no value provided, no label will be shown. You may use localization query strings to enable custom localization (for example, label: "@i18n:namespace.StringName"). Localization query strings allow you to use strings from any widget namespace or to create your own namespace in the localization file (i18n.json) and use strings from there (for example, label: "@i18n:myCustomNamespace.myCustomString"). For more information, see the Labels section.
wrapper	HTML string	" "	Each input exists in its own row in the form. By default this is a table-row with the label in the left cell and the input in the right cell. You can redefine this wrapper and layout by specifying a new HTML row structure. See the Wrappers section for more info. The default wrapper for an input is "
validate	function		Define a validation function for the input that executes when the input loses focus (blur) or changes value. Your function must return true or false. True to indicate it passed, false to indicate it failed. If your validation fails, the form will not submit and

Property	Type	Default	Description
			the invalid input will be highlighted in red. See the Validation section for more details and examples.
validateWhileTyping	boolean	false	Execute validation on keypress in addition to blur and change. This ignores non-character keys like shift, ctrl, and alt.
options	array	[]	When 'type' is set to 'select', you can populate the select by adding options to this array. Each option is an object (for example, {name: 'Option 1', value: '1'} for a selectable option, and {name: "Group 1", group: true} for an option group).

HTML attributes

With the exception of special properties, all properties will be added as HTML attributes on the input element. You can use standard HTML attributes or make your own.

Example

```
{
  id: "cx_callback_form_firstname",
  name: "firstname",
  maxlength: "100",
  placeholder: "@i18n:callback.CallbackPlaceholderOptional",
  label: "@i18n:callback.CallbackFirstName"
}
```

In this example, `id`, `name`, `maxlength`, and `placeholder` are all standard HTML attributes for the text input element. Whatever values are set here will be applied to the input as HTML attributes.

Note: the default input type is "text", so type does not need to be defined if you intend to make a text input.

HTML output

Labels

A label tag will be generated for your input if you specify label text and if your custom input wrapper includes a '{label}' designation. If you have added an ID attribute for your input, the label will

automatically be linked to your input so that clicking on the label selects the input or, for check boxes, toggles it.

Labels can be defined as static strings or localization queries.

Wrappers

Wrappers are HTML string templates that define a layout. There are two kinds of wrappers, **form wrappers** and **input wrappers**:

Form wrapper

You can specify the parent wrapper for the overall form in the top-level "wrapper" property. In the example below, we specify this value as ". This is the default wrapper for the Callback form.

```
{
    wrapper: "
", /* form wrapper */ inputs: [] }
```

Input wrapper

Each input is rendered as a table row inside the form wrapper. You can change this by defining a new wrapper template for your input row. Inside your template you can specify where you want the input and label to be by adding the identifiers "{label}" and "{input}" to your wrapper value. See the example below:

```
{
    id: "cx_callback_form_firstname",
    name: "firstname",
    maxlength: "100",
    placeholder: "@i18n:callback.CallbackPlaceholderOptional",
    label: "@i18n:callback.CallbackFirstName"
    wrapper: "{label}{input}" /* input row wrapper */
}
```

The {label} identifier is optional. Omitting it will allow the input to fill the row. If you decide to keep the label, you can move it to any location within the wrapper, such as putting the label on the right, or stacking the label on top of the input. You can control the layout of each row independently, depending on your needs.

You are not restricted to using a table for your form. You can change the form wrapper to "

" and then change the individual input wrappers from a table-row to your own specification. Be aware though that when you move away from the default table wrappers, you are responsible for styling and aligning your layout. Only the default table-row wrapper is supported by default Themes and CSS.

Validation

You can apply a validation function to each input that lets you check the value after a change has been made and/or the user has moved to a different input (on change and on blur). You can enable validation on key press by setting `validateWhileTyping` to true in your input definition.

Here is how a validation function is defined:

```
{
  id: "cx_callback_form_firstname",
  name: "firstname",
  maxlength: "100",
  placeholder: "@i18n:callback.CallbackPlaceholderOptional",
  label: "@i18n:callback.CallbackFirstName"

  validateWhileTyping: true, // default is false

  validate: function(event, form, input, label, $, CXBus, Common){
    if(input && input.val()) { // to validate some input exists in the
      // firstname input field (required field)
      return true;           // validation passed
    }else{
      return false;          // no input exists, validation failed
    }
  }
}
```

You can perform any validation you like in the validate function but it must return `true` or `false` to indicate that validation has passed or failed, respectively. If you return `false`, the Callback form will not submit, and the input will be highlighted in red. This is achieved by adding the CSS class `"cx-error"` to the input.

Validation function arguments

Argument	Type	Description
event	JavaScript event object	The input event reference object related to the form input field. This event data can be helpful to perform actions like active validation on an input field while the user is typing.
form	HTML reference	A jquery reference to the form wrapper element.
input	HTML reference	A jquery reference to the input element being validated.
label	HTML reference	A jquery reference to the label for the input being validated.
\$	jquery instance	Widget's internal jquery instance. Use this to help you write your validation logic, if needed.
CXBus	CXBus instance	Widget's internal CXBus reference. Use this to call commands on the bus, if needed.
Common	Function Library	Widget's internal Common library of functions and utilities. Use if needed.

Form submit

Custom input field form values are submitted to the server as key value pairs in the form submit request, where the input field names are the property keys and the input field values are the property values.

Form pre-fill

You can pre-fill the custom form using the `Callback.open` command by passing the form (form data) and formJSON (custom registration form), provided the form input names in the formJSON must match with the property names in the form data.

The following example will open the Callback form with the phone number already entered in the Phone input field.

```
_genesys.widgets.bus.command("Callback.open", {  
    formJSON: {  
        wrapper: "  
", inputs: [{ id: "cx_form_phone_number", name: "phonenumber", maxlength:  
"12", placeholder: "@i18n:callback.CallbackPlaceholderPhoneNumber", label:  
"@i18n:callback.CallbackPhoneNumber" }] }, form: { phonenumber:  
9453222222 } });
```

CallUs

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Customization](#)
 - [1.3 Namespace](#)
 - [1.4 Mobile support](#)
 - [1.5 Screenshots](#)
- [2 Configuration](#)
 - [2.1 Example](#)
 - [2.2 Options](#)
- [3 Localization](#)
 - [3.1 Usage](#)
 - [3.2 Example i18n JSON](#)
- [4 API commands](#)
 - [4.1 open](#)
 - [4.2 close](#)
 - [4.3 configure](#)
- [5 API events](#)

- Developer

Learn how to display an overlay screen showing one or more phone numbers for customer service, as well as the hours that this service is available.

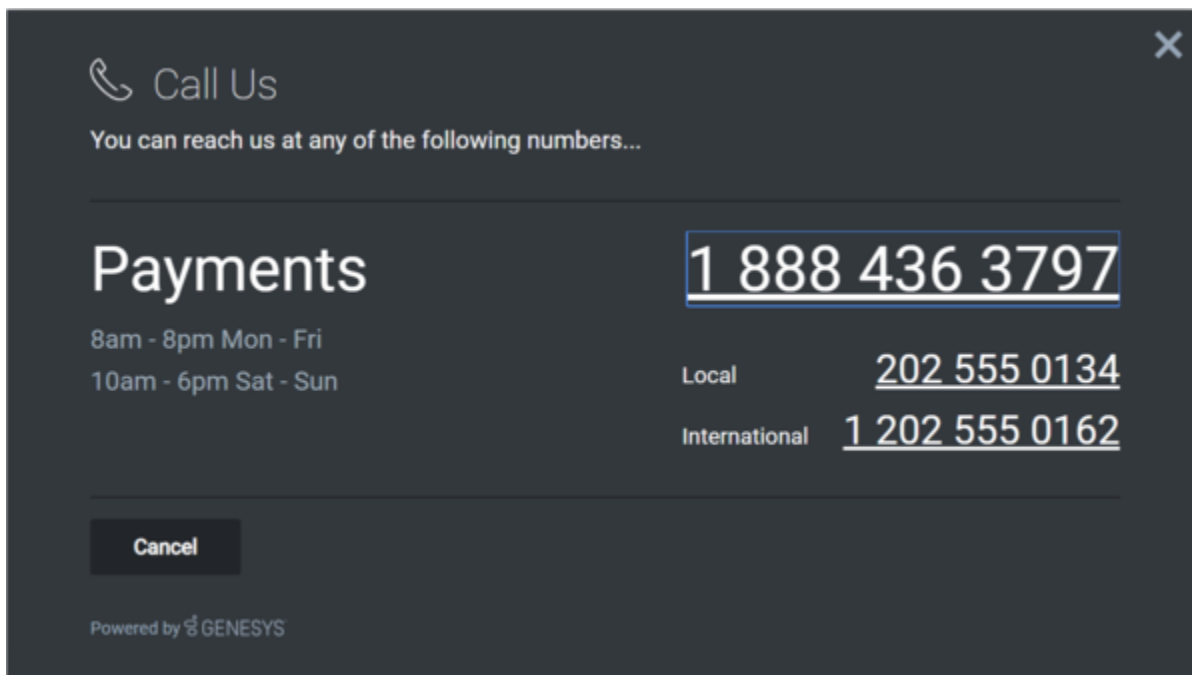
Related documentation:

-

[Link to video](#)

Overview

The CallUs Widget provides an overlay screen showing one or more phone numbers for customer service, as well as the hours that this service is available. The arrangement of numbers in this layout starts with a main phone number, which can be followed by alternative or additional phone numbers. Each number can be named, and there is no limit to the number of phone numbers you can include. If the list of numbers doesn't fit in the widget, the user can scroll down to see the rest.



Usage

Launch CallUs manually by using the following methods:

- Call the **CallUs.open** command
- Configure ChannelSelector to show CallUs as a channel
- Create your own custom button or link to open CallUs (using the "CallUs.open" command)

Important

By default a user has no way of launching the CallUs Widget. You must choose a suitable method for launching this widget.

Customization

You can customize and localize all of the text, titles, names, and numbers shown in the CallUs Widget by adding entries into your configuration and localization options. There are no formatting requirements. Text will appear as you entered it.

Important

If you do not configure the CallUs Widget it will appear as an empty overlay. You must configure this Widget before using it.

CallUs supports themes. You can create and register your own themes for Genesys Widgets.

Namespace

The CallUs plugin has the following namespaces tied up with each of the following types:

Type	Namespace
Configuration	callus
i18n—Localization	callus
CXBus—API commands & API events	CallUs
CSS	.cx-call-us

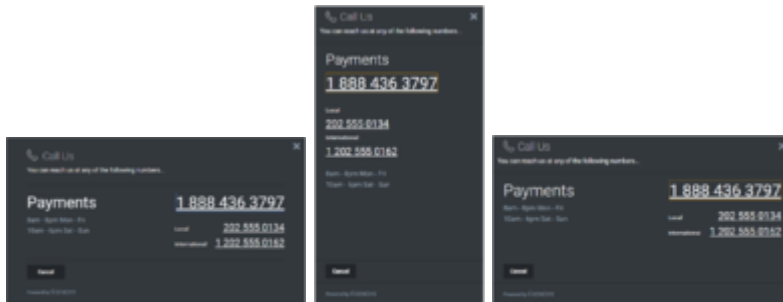
Mobile support

CallUs supports both desktop and mobile devices. Like all Genesys Widgets, there are two main modes: Desktop & Mobile. Desktop is employed for monitors, laptops, and tablets. Mobile is employed for smartphones. When a smartphone is detected, CallUs switches to special full-screen templates that are optimized for both portrait and landscape orientations.

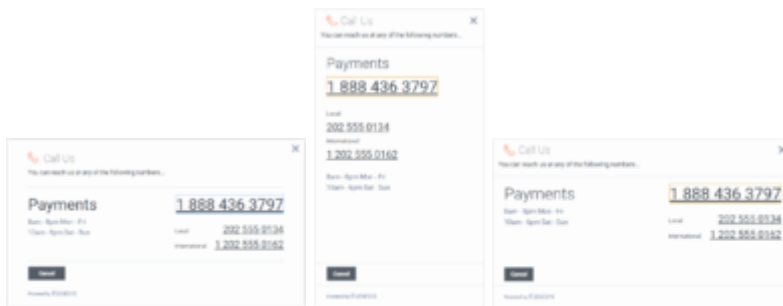
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

Dark theme



Light theme



Configuration

CallUs uses the **`_genesys.widgets.callus`** configuration property. You must specify all of the numbers and labels that appear in the CallUs UI.

Example

```
window._genesys.widgets.callus = callus: {
  contacts: [
    {
      displayName: 'Payments',
      i18n: 'Number001',
      number: '1 202 555 0162'
    },
    {
      displayName: 'Local',
      i18n: 'Number002',
      number: '202 555 0134'
    }
  ]
}
```

```

        displayName: 'International',
        i18n: 'Number003',
        number: '0647 555 0131'
    },
    hours: [
        '8am - 8pm Mon - Fri',
        '10am - 6pm Sat - Sun'
    ]
};

```

Options

Name	Type	Description	Default	Required
contacts	array	<p>An array of objects that represent phone numbers and their labels. The first number in this list will display as the larger, main number. Phone labels can be set directly using the 'displayName' property or you can use String Names from your localization file by setting the String Name in the 'i18n' property. 'i18n' overrides 'displayName'.</p> <p>Example</p> <pre>{ "displayName": "Payments", "i18n": "Number001", "number": "1 202 555 0162" }</pre>	[]	true
hours	array	<p>Array of strings to show stacked in the business hours section. Strings here are freeform. See screenshots for ideas.</p>	[]	

Localization

Important

For information on how to set up localization, please refer to [Localize widgets and services](#).

Usage

Use the **callus** namespace when defining localization strings for the CallUs plugin in your i18n JSON file.

The following example shows how to define new strings for the **en** (English) language. You can use any language codes you wish; there is no standard format. When selecting the active language in your configuration, you must match one of the language codes defined in your i18n JSON file. Please note that you must only define a language code once in your i18n JSON file. Inside each language object you should define new strings for each widget.

Example i18n JSON

```
{
  "en": {
    "callus": {
      "CallUsTitle": "Call Us",
      "SubTitle": "You can reach us at any of the following NUMBERS...",
      "CancelButtonText": "Cancel",
      "AriaWindowLabel": "Call Us Window",
      "AriaCallUsClose": "Call Us Close",
      "AriaBusinessHours": "Business Hours",
      "AriaCallUsPhoneApp": "Opens the phone application",
      "AriaCancelButtonText": "Cancel"
    }
  }
}
```

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see [Genesys Widgets Extensions](#) for more information about extending Genesys Widgets.

CallUs

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('CallUs.open');
```

open

Opens the CallUs UI.

Example

```
oMyPlugin.command('CallUs.open').done(function(e){  
    // CallUs opened successfully  
}).fail(function(e){  
    // CallUs failed to open  
});
```

Resolutions

Status	When	Returns
resolved	CallUs is successfully opened	n/a
rejected	CallUs is already open	'Already opened'

close

Closes the CallUs UI.

Example

```
oMyPlugin.command('CallUs.close').done(function(e){  
    // CallUs closed successfully  
}).fail(function(e){  
    // CallUs failed to close  
});
```

Resolutions

Status	When	Returns
resolved	CallUs successfully closed	n/a
rejected	CallUs is already closed	'Already closed'

configure

Internal use only. The main App plugin shares configuration settings to widgets using each widget's configure command. The configure command can only be called once at startup. Calling configure again after startup may result in unpredictable behavior.

Example

```
oMyPlugin.command('CallUs.configure', {
  contacts: [
    {
      displayName: 'Payments',
      i18n: 'Number001',
      number: '1 888 436 3797'
    }
  ],
  hours: ['8am - 8pm Mon - Fri']
}).done(function(e){
  // CallUs configred successfully
}).fail(function(e){
  // CallUs failed to configure
});
```

Options

Option	Type	Description
contacts	Array	An array of objects that represent phone numbers and their labels. The first number in this list will display as the larger, main number.
hours	Array	Array of strings to show stacked in the business hours section. Strings here are freeform.

Resolutions

Status	When	Returns
resolved	CallUs configuration is provided	n/a
rejected	No configuration is provided	'Invalid Configuration'

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events.

Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see [Genesys Widgets Extensions](#) for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('CallUs.ready', function(e){});
```

Name	Description	Data
ready	CallUs is initialized and ready to accept commands	
opened	CallUs UI has been opened	
closed	CallUs UI has been closed	

ChannelSelector

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Customization](#)
 - [1.3 Namespace](#)
 - [1.4 Mobile support](#)
 - [1.5 Screenshots](#)
- [2 Configuration](#)
- [3 Example](#)
 - [3.1 Options](#)
- [4 Localization](#)
 - [4.1 Usage](#)
 - [4.2 Example i18n JSON](#)
- [5 API commands](#)
 - [5.1 close](#)
 - [5.2 open](#)
 - [5.3 configure](#)
 - [5.4 displayStats](#)
 - [5.5 disableStats](#)
 - [5.6 enableStats](#)
- [6 API events](#)

- Developer

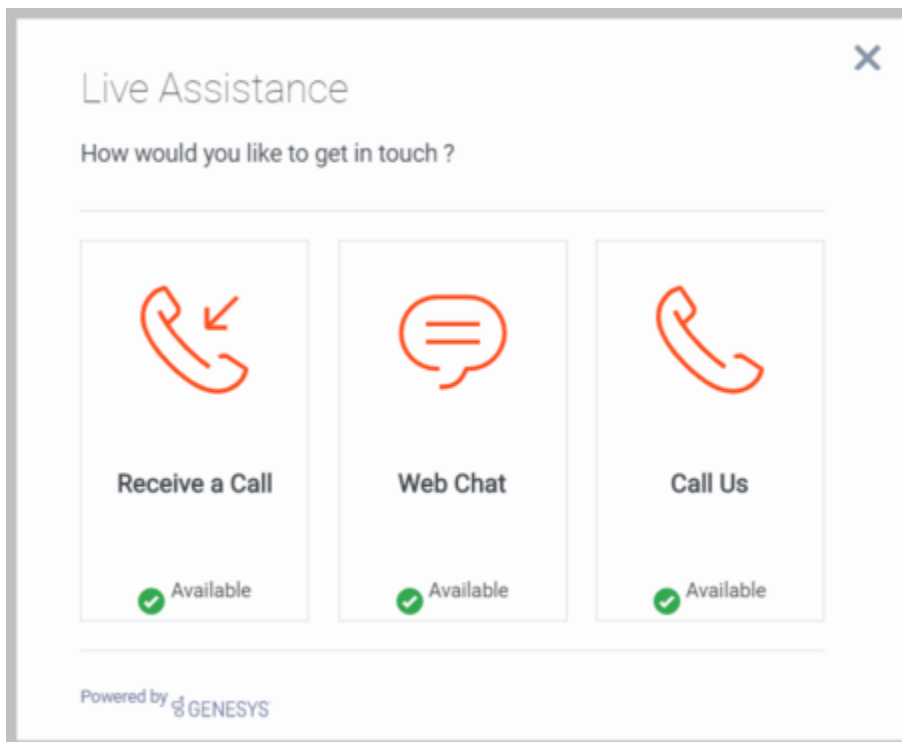
Learn how to provide your customers with a configurable list of channels as an entry point for contacting customer service.

Related documentation:

-

Overview

The ChannelSelector Widget displays a configurable list of channels, as an entry point for customers to contact customer service. In addition to showing multiple channels, ChannelSelector can be configured to display the estimated wait time (EWT) for each channel. You can also use an EWT value to configure channels to hide, or to show, that they disabled. Channels are not limited to Genesys Widgets; you can add your own custom channels to launch applications or open new windows as necessary.



Note the screenshots in the following section, and visit the configuration section for more information.

Usage

Use the following methods to launch ChannelSelector manually:

- Call the **ChannelSelector.open** command
- Create your own custom button or link to open ChannelSelector (using the "ChannelSelector.open" command)

Important

By default ChannelSelector has no channels configured. The UI will appear empty if not configured. See the configuration section for examples and information on how to set up your own custom channels.

Customization

You can customize and localize all of the static text shown in the ChannelSelector Widget by adding entries into your configuration and localization options.

ChannelSelector supports Themes. You can create and register your own themes for Genesys Widgets.

Namespace

The Channel Selector plugin has the following namespaces tied to each of the following types:

Type	Namespace
Configuration	channelselector
i18n—Localization	channelselector
CXBus—API commands & API events	ChannelSelector
CSS	.cx-channel-selector

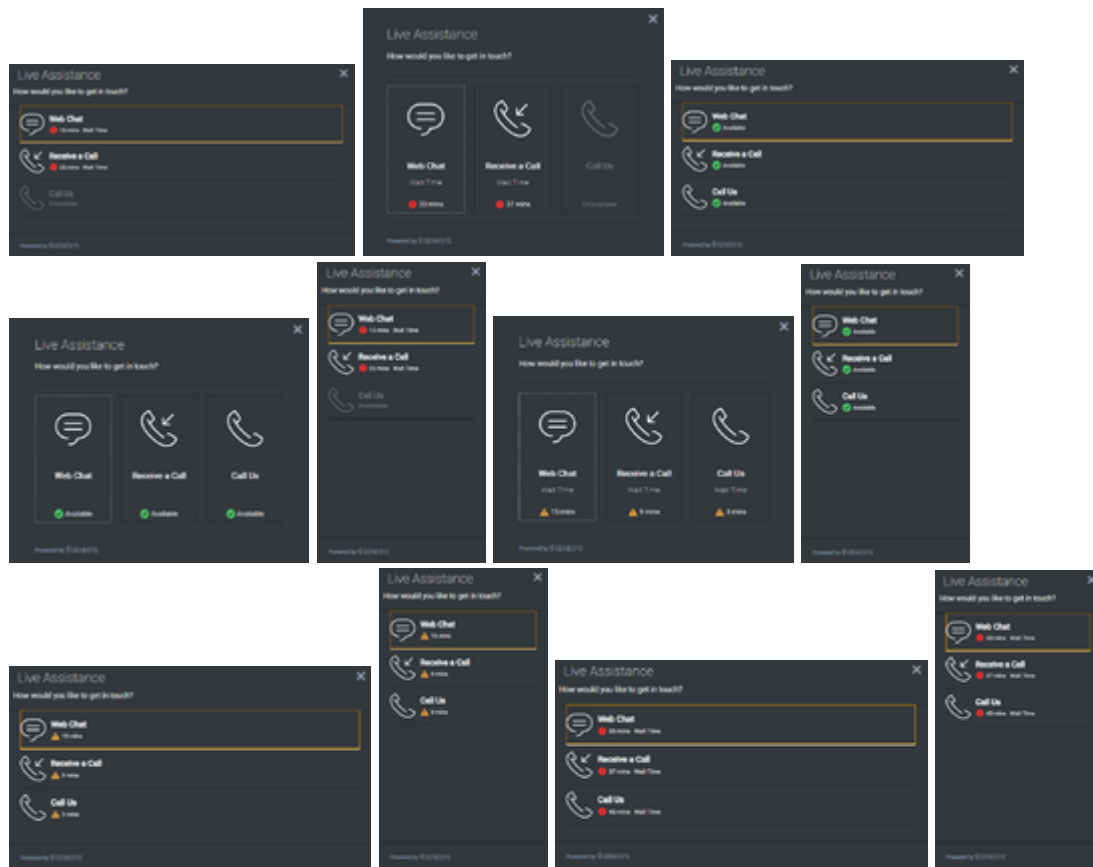
Mobile support

ChannelSelector supports both desktop and mobile devices. Like all Genesys Widgets, there are two main modes: Desktop and Mobile. Desktop is employed for monitors, laptops, and tablets. Mobile is employed for smartphones. When a smartphone is detected, ChannelSelector switches to special full-screen templates that are optimized for both portrait and landscape orientations.

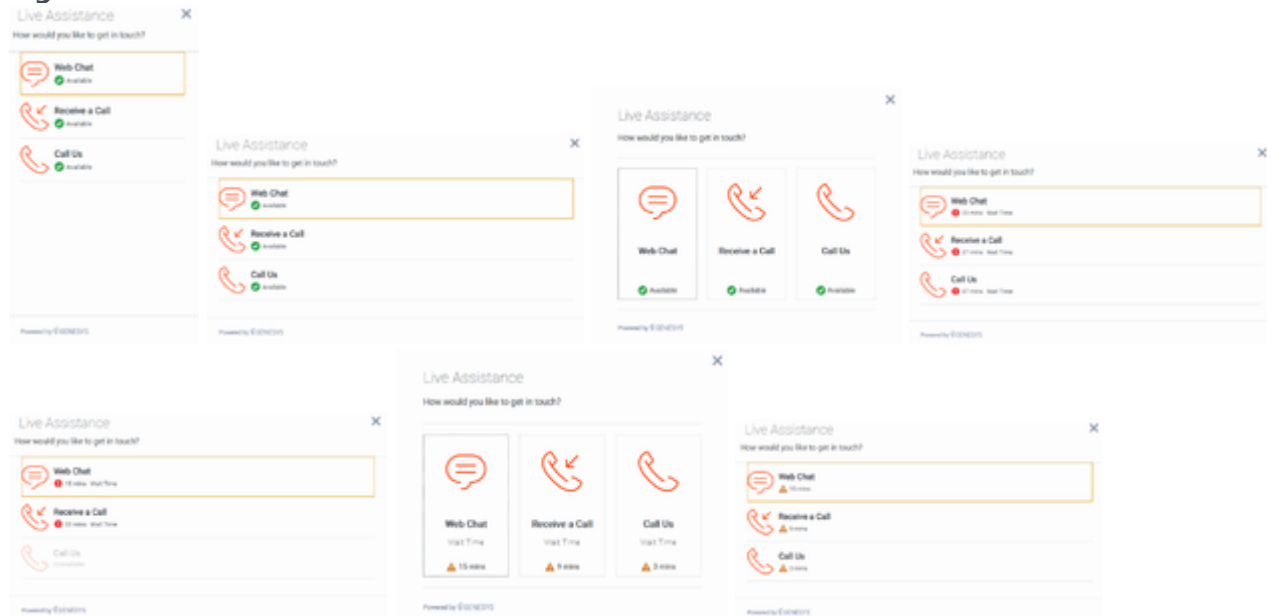
Switching between Desktop and Mobile modes is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile modes manually if necessary.

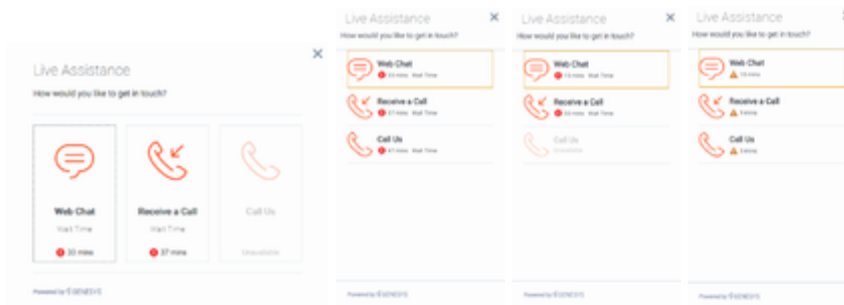
Screenshots

Dark theme



Light theme





Configuration

ChannelSelector shares the **_genesys.widgets.channelselector** configuration namespace. ChannelSelector has UI options to enable and disable channels, hide channels, add new channels, and display estimated wait time (EWT) details. All the channels are displayed based on the array of objects order defined in the channel's configuration. To hide a particular channel, simply remove the corresponding array object.

Important

EWT can only be configured for WebChat, Callback, ClickToCall, and CallUs channels. It may not be applicable for other channels. If configured for the Send Message channel, it will always be shown as available regardless of any EWT value.

Example

```
window._genesys.widgets.channelselector = {
  ewtRefreshInterval: 10,
  channels: [{
    enable: true,
    clickCommand: 'CallUs.open',
    displayName: 'Call Us',
    idn: 'CallusTitle',
    icon: 'call-outgoing',
    html: '',
    ewt: {
      display: true,
      queue: 'callus_ewt_test_eservices',
      availabilityThresholdMin: 300,
      availabilityThresholdMax: 480,
      hideChannelWhenThresholdMax: false
    }
  },
  {
```



```
enable: true,
clickCommand: 'WebChat.open',
displayName: 'Web Chat',
i18n: 'ChatTitle',
icon: 'chat',
html: '',
ewt: {
  display: true,
  queue: 'chat_ewt_test_eservices',
  availabilityThresholdMin: 300,
  availabilityThresholdMax: 480,
  hideChannelWhenThresholdMax: false
},
{
  enable: true,
  clickCommand: 'Callback.open',
  displayName: 'Receive a Call',
  i18n: 'CallbackTitle',
  icon: 'call-incoming',
  html: '',
  ewt: {
    display: true,
    queue: 'callback_ewt_test_eservices',
    availabilityThresholdMin: 300,
    availabilityThresholdMax: 480,
    hideChannelWhenThresholdMax: false
  }
},
};
```

Options

Name	Type	Description	Default	Required
ewtRefreshInterval	number	EWT is updated for every time interval (seconds) defined here.	10	n/a
channels[].enable	boolean	Enable/disable a channel.	true	n/a
channels[].clickCommand	string	The CXBus command name for opening a particular Widget when this channel is clicked.	none	Always
channels[].displayName	string	A channel name to display on ChannelSelector Widget.	none	Always
channels[].i18n	string	To support localization of the	none	n/a

Name	Type	Description	Default	Required
		channel display name, this takes a key parameter of the channelselector section in the language pack file. Overrides above displayName.		
channels[].icon	string	Select from one of the Genesys Widgets icons by specifying icon css class name.	none	Always
channels[].html	string	Overrides and replaces the icon section of a channel with the html (image tag) defined here.	none	n/a
channels[].ewtdisplay	boolean	To display EWT details.	true	n/a
channels[].ewt.queue	string	EWT service channel virtual queue.	none	Always
channels[].ewt.availabilityThreshold	integer (seconds)	<p>If EWT is greater than 0 minutes and less than this minimum threshold value (in minutes), then the EWT is shown with a yellow warning icon.</p> <div> Important Comparison is made after converting the threshold value in seconds to minutes. </div>	300	n/a
channels[].ewt.availabilityThresholdMax	integer (seconds)	<p>If EWT is greater than this minimum threshold value (in minutes) and less than the maximum threshold value (in minutes), then the EWT is shown with a red alert icon.</p> <div> Important Comparison is made </div>	480	n/a

Name	Type	Description	Default	Required
		after converting the threshold value in seconds to minutes.		
channels[].ewt.hideChannelWhenThresholdIsGreater	boolean	Hides this channel when EWT is greater than the maximum threshold value.	true	n/a

Localization

Important

For information on how to set up localization, refer to [Localize widgets and services](#).

Usage

Use the **channelselector** namespace when you define localization strings for the ChannelSelector plugin in your i18n JSON file.

The following example shows how to define new strings for the **en** (English) language. You can use any language codes you wish; there is no standard format. When selecting the active language in your configuration, you must match one of the language codes defined in your i18n JSON file. You must only define a language code once in your i18n JSON file. Inside each language object you must define new strings for each Widget.

Example i18n JSON

```
{
  "en": {
    "channelselector": {
      "Title": "Live Assistance",
      "SubTitle": "How would you like to get in touch?",
      "WaitTimeTitle": "Wait Time",
      "AvailableTitle": "Available",
      "AriaAvailableTitle": "Available",
      "UnavailableTitle": "Unavailable",
      "CallbackTitle": "Receive a Call",
      "AriaClose": "Live Assistance Close",
      "AriaWarning": "Warning",
      "AriaAlert": "Alert",
      "minute": "min",
      "minutes": "mins",
      "AriaWindowLabel": "Live Assistance Window"
    }
  }
}
```

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('ChannelSelector.open');
```

close

Closes the ChannelSelector UI.

Example

```
oMyPlugin.command('ChannelSelector.close').done(function(e){  
    // ChannelSelector closed successfully  
}).fail(function(e){  
    // ChannelSelector failed to close  
});
```

Resolutions

Status	When	Returns
resolved	ChannelSelector is successfully closed	n/a
rejected	ChannelSelector is already closed	Already closed

open

Opens the ChannelSelector UI.

Example

```
oMyPlugin.command('ChannelSelector.open').done(function(e){  
    // ChannelSelector opened successfully
```

```
}).fail(function(e){  
    // ChannelSelector failed to open  
});
```

Resolutions

Status	When	Returns
resolved	ChannelSelector Widget is successfully opened	n/a
rejected	ChannelSelector Widget is already open	'Already open'

configure

Modifies the ChannelSelector configuration.

Example

```
oMyPlugin.command('ChannelSelector.configure', {  
    channels: [  
        {  
            enabled: true,  
            clickCommand: 'CallUs.open',  
            readyEvent: 'CallUs.ready',  
            displayName: 'Call Us',  
            id: 'CallusTitle',  
            icon: 'call-outgoing',  
            html: '',  
            ewt:{  
  
                display: true,  
                queue:'chat_ewt_test_eservices',  
                availabilityThresholdMin:60,  
                availabilityThresholdMax:600  
            }  
        }  
    ]  
}).done(function(e){  
    // ChannelSelector configured successfully  
}).fail(function(e){  
    // ChannelSelector failed to configure  
});
```

Options

Option	Type	Description
ewtRefreshInterval	number	EWT is updated for every time interval (seconds) is defined.

Option	Type	Description
channels	array	Array containing each channel configuration object. The order of channels is displayed based on the order defined here.
channels[].enable	boolean	Enable/disable chat channel.
channels[].clickCommand	string	The CXBus command name for opening a particular Widget when this channel is clicked.
channels[].readyEvent	string	Subscribes to this ready event published by a plugin and enables the channel when that plugin is ready.
channels[].displayName	string	A channel name to display in the ChannelSelector Widget.
channels[].i18n	string	To support localization of channel display name, this takes a key parameter of the channelselector section in the language pack file. Overrides above displayName.
channels[].icon	string	Select from one of the Genesys Widgets icons by specifying icon css class name.
channels[].html	string	Overrides and replaces the icon section of a channel with the html (image tag) defined here.
channels[].ewt.display	boolean	To display EWT details.
channels[].ewt.queue	string	EWT service channel virtual queue name.
channels[].ewt.availabilityThresholdMin	Number (seconds)	<p>If EWT is greater than 0 minutes and less than this minimum threshold value (in minutes), then the EWT is shown with a yellow warning icon.</p> <p>Note: Comparison is made after converting the threshold value in seconds to minutes.</p>
channels[].ewt.availabilityThresholdMax	Number (seconds)	<p>If EWT is greater than this minimum threshold value (in minutes) and less than the maximum threshold value (in minutes), then the EWT is shown with a red alert icon.</p> <p>Note: Comparison is made after converting the threshold value in seconds to minutes.</p>

Option	Type	Description
channels[].ewt.hideChannelWhenThresholdExceeded	boolean	Hides this channel when EWT is greater than the maximum threshold value.

Resolutions

Status	When	Returns
resolved	Configuration options are provided and set	n/a
rejected	No configuration options are provided	'Invalid configuration'

displayStats

Displays estimated wait time (EWT) and availability details for each channel.

Example

```
oMyPlugin.command('ChannelSelector.displayStats').done(function(e){
    // ChannelSelector displayed stats successfully
}).fail(function(e){
    // ChannelSelector failed to display stats
});
```

Resolutions

Status	When	Returns
resolved	EWT is displayed successfully	n/a
rejected	StatsService fails to retrieve EWT data	'Unable to display EWT Stats in ChannelSelector'
rejected	enableEwt config is disabled or when required channel plugins are not ready	'Either EWT config is disabled or plugins not yet ready'

disableStats

Clears the UI of any EWT. Disables EWT fetching for the defined time interval.

Example

```
oMyPlugin.command('ChannelSelector.disableStats').done(function(e){  
    // ChannelSelector disabled stats successfully  
}).fail(function(e){  
    // ChannelSelector failed to disable stats  
});
```

Resolutions

Status	When	Returns
resolved	ChannelSelector Widget is successfully opened	n/a
rejected	ChannelSelector Widget is not opened	'ChannelSelector not opened to disable stats details'
rejected	EWT is disabled for all channels	'Stats already disabled'

enableStats

Displays EWT and availability details in the UI. Enables fetching EWT for the defined time interval.

Example

```
oMyPlugin.command('ChannelSelector.enableStats').done(function(e){  
    // ChannelSelector enabled stats successfully  
}).fail(function(e){  
    // ChannelSelector failed to enable stats  
});
```

Resolutions

Status	When	Returns
resolved	ChannelSelector Widget is successfully opened	n/a
rejected	EWT details are already displayed	'Stats already enabled'
rejected	ChannelSelector Widget is not opened	'ChannelSelector not opened to enable stats details'

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('ChannelSelector.ready', function(e){});
```

Name	Description	Data
ready	ChannelSelector plugin is initialized and ready to accept commands	n/a
opened	ChannelSelector Widget has appeared on screen	n/a
closed	ChannelSelector Widget has been removed from the screen	n/a

Console

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
- [2 Configuration](#)
 - [2.1 Description](#)
 - [2.2 Example](#)
 - [2.3 Options](#)
- [3 Localization](#)
- [4 Strings](#)
- [5 API commands](#)
 - [5.1 open](#)
 - [5.2 close](#)
 - [5.3 configure](#)
- [6 API events](#)

- Developer

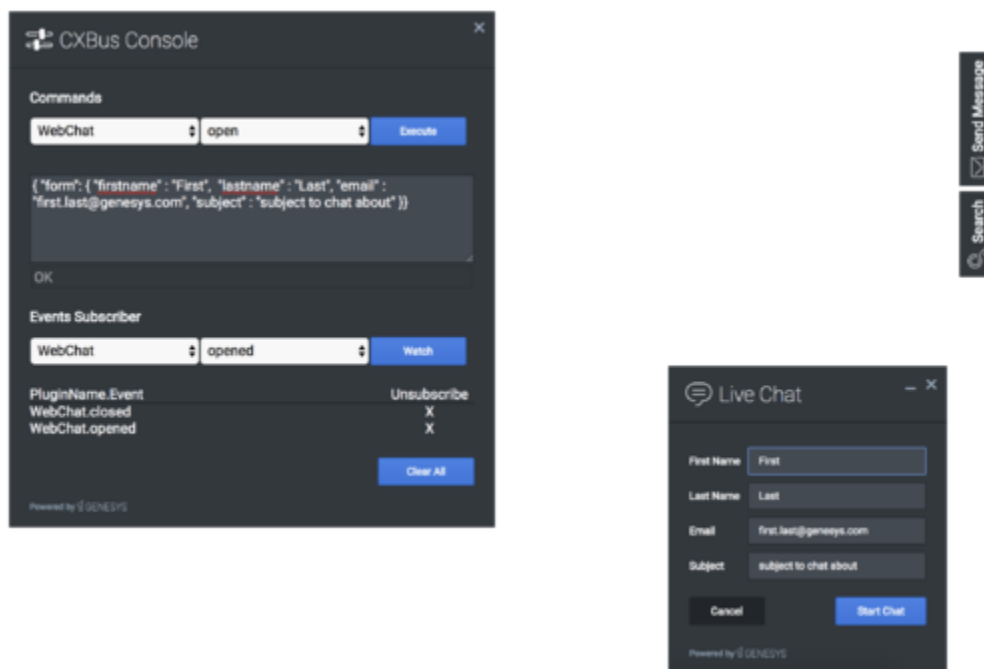
Learn how to debug commands and events on the widget bus.

Related documentation:

-

Overview

Use the Console Widget to debug commands and events on the widget bus. You can use dynamically populated lists to test, debug, or demo all commands. You can also create event watch lists that alert you when an event has fired.



Console provides an easy-to-use interface for debugging the widget bus that complements the standard command-line methods. You can drag and drop the console anywhere on your screen, and when you refresh the page or move to another one, Console reappears right where you left it. It is a great tool for getting to know the widget bus, the API for each widget, and debugging issues.

Usage

Launch WebChat manually by using the following methods:

- Call the **Console.open** command
- Configure the settings to show Console when the browser window is opened.
- Create your own custom button or link to open Console (using the **Console.open** command)

Configuration

Description

Console option to open on initial loading.

Example

```
window._genesys.widgets.console = {open: true};
```

Options

Name	Type	Description	Default	Required
open	boolean	Set to true for console to open at start.	false	false

Localization

Important

For information on how to set up localization, please refer to [Localize widgets and services](#).

Strings

```
{
  "ConsoleTitle": "CXBUS Console",
  "Commands": "Commands",
  "Plugin": "Plugin",
  "ConsoleErrorButton": "OK",
  "Execute": "Execute",
  "Event": "Event",
  "SubscribeTo": "Subscribe to",
  "Unsubscribe": "Unsubscribe",
  "ReturnData": "Return Data",
  "EventsSubscriber": "Events Subscriber",
```

```
"Watch": "Watch",
"pluginNameEvent": "PluginName.Event",
"ClearAll": "Clear All",
"OptionsSample": "JSON Formatted Options {'option': value}"
}
```

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');
oMyPlugin.command('Console.open');
```

open

Opens the Console UI.

Example

```
oMyPlugin.command('Console.open').done(function(e){
    // Console opened successfully
}).fail(function(e){
    // Console failed to open
});
```

Resolutions

Status	When	Returns
resolved	Console is successfully opened	n/a
rejected	Console is already open	'Already opened'

close

Closes the Console UI.

Example

```
oMyPlugin.command('Console.close').done(function(e){  
    // Console closed successfully  
}).fail(function(e){  
    // Console failed to close  
});
```

Resolutions

Status	When	Returns
resolved	Console successfully closed	n/a
rejected	Console is already closed	'Already closed'

configure

Modifies the Console configuration options. See the Console configuration page.

Example

```
oMyPlugin.command('Console.configure', {  
    open: false  
}).done(function(e){  
    // Console configured successfully  
}).fail(function(e){  
    // Console failed to configure  
});
```

Options

Option	Type	Description
open	boolean	If setting is open: true, the console will automatically be open when Widgets is launched and the console is ready.

Resolutions

Status	When	Returns
resolved	Console configuration is provided	n/a
rejected	No configuration is provided	'Invalid Configuration'

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see [Genesys Widgets Extensions](#) for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('Console.ready', function(e){});
```

Name	Description	Data
ready	Console is initialized and ready to accept commands.	n/a
opened	The Console Widget has appeared on screen.	n/a
closed	The Console Widget has been removed from the screen.	n/a

SideBar

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Dependency](#)
 - [1.3 Customization](#)
 - [1.4 Namespace](#)
 - [1.5 Mobile support](#)
 - [1.6 Screenshots](#)
- [2 Configuration](#)
 - [2.1 Example](#)
 - [2.2 Options](#)
- [3 Localization](#)
 - [3.1 Strings](#)
- [4 API commands](#)
 - [4.1 configure](#)
- [5 API events](#)
 - [5.1 Resolutions](#)
 - [5.2 open](#)
 - [5.3 close](#)
 - [5.4 expand](#)
 - [5.5 contract](#)

- Developer

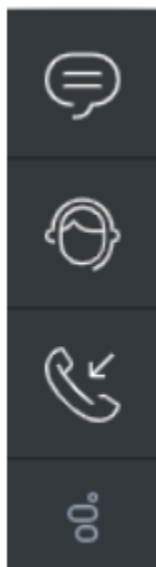
Learn about the Sidebar widget, which customers use to launch other widgets with a single click.

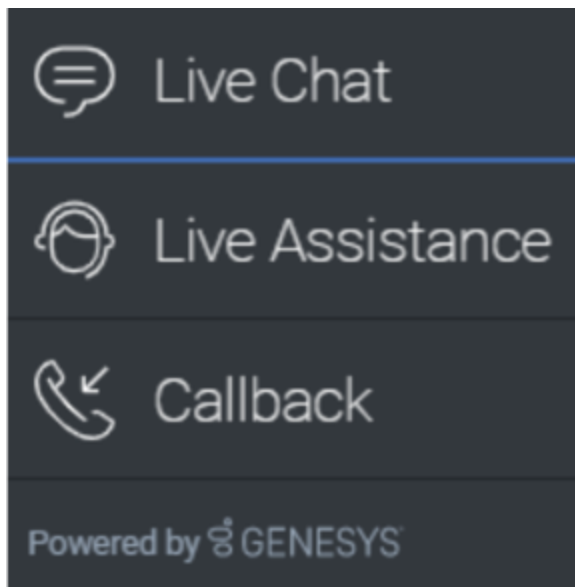
Related documentation:

-

Overview

Use the Sidebar to launch other widgets with a single click. By default, Sidebar is displayed on the right side of the screen, and you can configure any launchable widgets onto Sidebar, including your custom extension widgets. The Sidebar UI expands when you hover your cursor over it, and contracts when you move the cursor away. Other features include configurable positioning and mobile support. You can also add new configurations on the fly, which automatically re-renders the sidebar.





Usage

Use the following methods to launch SideBar manually:

- Call the **SideBar.open** command
- Configure Sidebar to show and launch custom widgets.

Dependency

You must configure at least one customer-facing UI widget in order to use the Sidebar Widget.

Customization

You can customize and localize all of the text shown in the Sidebar Widget by adding entries to your configuration and localization options.

Sidebar supports themes. You can create and register your own themes for Genesys Widgets.

Namespace

The Sidebar plugin has the following namespaces tied up with each of the following types:

Type	Namespace
Configuration	sidebar
i18n—Localization	sidebar
CXBus—API commands & API events	SideBar
CSS	.cx-sidebar

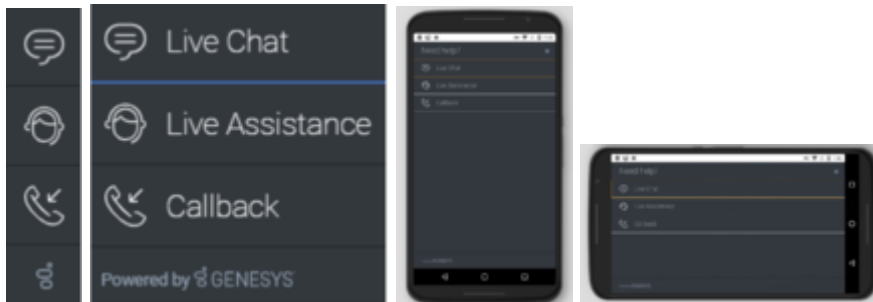
Mobile support

SideBar supports both desktop and mobile devices. In mobile mode, the sidebar launcher button is displayed to the bottom of the screen. When triggered, it expands to the full screen of mobile and shows all channels configured with scrollbar when necessary. Like all Genesys Widgets, there are two main modes: Desktop & Mobile. Desktop is employed for monitors, laptops, and tablets. Mobile is employed for smartphones. When a smartphone is detected, SideBar switches to special full-screen templates that are optimized for both portrait and landscape orientations.

Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

Dark theme



Light theme



Configuration

SideBar shares the **`_genesys.widgets.sidebar`** configuration namespace. SideBar has UI options to handle its position on the screen, disable expand feature sidebar, hide sidebar, and add new channels on the fly. The display order of channels is based on the order defined in channels configuration array.

Example

```
window._genesys.widgets.sidebar = {
    showOnStartup: true,
    position: 'left',
    expandOnHover: true,
    channels: [{
        name: 'ChannelSelector',
        clickCommand: 'ChannelSelector.open',
        clickOptions: {},
        //use your own static string or i18n query string for the below two
        displayName: 'Live Assist',
        displayTitle: 'Get live help',
        icon: 'agent'
    },
    {
        name: 'WebChat'
    }
    ]
};
```

Options

Name	Type	Description	Default	Required
showOnStartup	boolean	Shows the sidebar on the screen when Widgets is launched.	true	false
position	string	Defines the position of sidebar on the screen. Acceptable values are left or right.	right	false
expandOnHover	boolean	Enables the expand (slide-out) or contract (slide-in) behavior of sidebar.	true	false
channels[index].name	string	Name of the channel. It can be found in the namespace section documentation of each Widget. Used to identify official channels vs	n/a	true

Name	Type	Description	Default	Required
		custom channels. If a reserved name is used here, Sidebar will apply default values for that channel. A plugin name defined in the new custom plugin can also be given here. To override the default values or when defining a new custom channel/plugin, use the below following properties.		
channels[index].clickCommand	string	Change the default command that is triggered when clicked.	n/a	false
channels[index].clickOptions	Object	Pass valid command options that are used in above click command execution.	n/a	n/a
channels[index].readyEvent	string	Subscribes to this ready event published by a plugin.	n/a	false
channels[index].displayName	string or i18n query string	Change the default display name for this channel with your own static string or to achieve localization, use i18n query string. Syntax: @i18n:.	n/a	false
channels[index].displayTitle	string or i18n query string	Change the default tooltip content for this channel with your own static string or to achieve localization, use i18n query string. Syntax: @i18n:.	n/a	false
channels[index].icon	string	Change the default icon for this channel. For the	n/a	false

Name	Type	Description	Default	Required
		list of icon names see <i>Customize icons</i> in Customize appearance.		
channels[index].onClick	function	Define a custom onclick function, this overrides clickCommand and clickOptions.	n/a	false

Localization

For your custom plugins, you can define string key names and values for Name and Title (tooltip) to display on sidebar. The key format requires the plugin name, followed by "Title" or "Name". For example, a plugin named "MyPlugin" will have keys called "MyPluginName" and "MyPluginTitle".

Important

For information on how to set up localization, refer to [Localize widgets and services](#).

Strings

```
{
  "SidebarTitle": "Need help?",
  "ChannelSelectorName": "Live Assistance",
  "ChannelSelectorTitle": "Get assistance from one of our agents right away",
  "CallUsName": "Call Us",
  "CallUsTitle": "Call Us details",
  "CallbackName": "Callback",
  "CallbackTitle": "Receive a Call",
  "WebChatName": "Live Chat",
  "WebChatTitle": "Live Chat",
  "AriaClose": "Close the menu Need help"
}
```

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');

oMyPlugin.command('SideBar.open');
```

configure

Internal use only. The main App plugin shares configuration settings to widgets using each widget's configure command. Sidebar widget has to be configured at least with one channel. The configure command can also be called at runtime with new configuration, this will override the existing configuration showing new channels on the screens.

Example

```
oMyPlugin.command('SideBar.configure', {
    showOnStartup: false,
    position: 'left',
    expandOnHover: false,
    channels: [
        {
            name: 'ChannelSelector',
            clickCommand: 'ChannelSelector.open',
            clickOptions: {},

            //use your own static string or i18n query string for the below two
            //display properties.
            //Example for i18n query string: '@i18n:sidebar.ChannelSelectorName' where 'sidebar' refers to
            //plugin namespace and
            //ChannelSelectorName' name refers to the property key containing the actual text.
            displayName: '@i18n:sidebar.ChannelSelectorName',
            displayTitle: 'Get assistance from one of our agents right away', //
            //Your own static string
            readyEvent: 'ChannelSelector.ready',
            icon: 'agent',
            onClick: function($, CXBus, Common) {
                _genesys.widgets.bus.command('MyPlugin.open');
            }
        },
        ...
    ]
}).done(function(e){
    // Sidebar configured successfully
}).fail(function(e){
    // Sidebar failed to configure properly
});
```

Options

Option	Type	Description
showOnStartup	boolean	Shows the sidebar on the screen when Widgets is launched.
position	string	Defines the position of sidebar on the screen.
expandOnHover	boolean	Enables the expand or contract behavior of sidebar.
channels	array	Array containing each channel configuration object. The order of channels are displayed based on the order defined here.
channels[index].name	string	Name of the channel. It can be found in the namespace section documentation of each Widget. Used to identify official channels vs custom channels. If a reserved name is used here, Sidebar will apply default values for that channel. To override the default values or when defining a new custom channel, use the below following properties.
channels[index].clickCommand	string	Change the default command that is triggered when clicked.
channels[index].clickOptions	object	Pass valid command options that are used in above click command execution.
channels[index].displayName	string or i18n query string	Change the default display name for this channel with your own static string or to achieve localization, use i18n query string. Syntax: @i18n:..
channels[index].displayTitle	string or i18n query string	Change the default tooltip content for this channel with your own static string or to achieve localization, use i18n query string. Syntax: @i18n:..
channels[index].readyEvent	string	Subscribes to this ready event published by a plugin.
channels[index].icon	string	Change the default Icon for this channel. For the list of Icon names see <i>Customize icons</i> in <i>Customize appearance</i> .
channels[index].onClick	function	Define a custom onclick function, this overrides clickCommand and clickOptions.

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('SideBar.ready', function(e){ /* sample code */ });
```

Name	Description	Data
ready	Sidebar is initialized and ready to accept commands.	n/a
opened	Sidebar widget has appeared on screen. For desktop it is displayed on the sides of the screen and in mobiles at the bottom corner as a button.	n/a
closed	Sidebar widget has been removed from the screen.	n/a
expanded	Sidebar widget has expanded, showing channel icon and name.	n/a
contracted	Sidebar widget has contracted back, showing channel icons only.	n/a

Resolutions

Status	When	Returns
resolved	When configuration options are provided and set	n/a
rejected	When no configuration options are provided	'Invalid configuration. Please ensure at least one channel is configured.'

open

Opens the SideBar UI. In Desktop mode, it opens as an actual SideBar and shows the configured channels where as in mobile it opens as a button at the bottom to start.

Example

```
oMyPlugin.command('SideBar.open');
```

Resolutions

Status	When	Returns
resolved	When sidebar is successfully opened	n/a
rejected	When sidebar is already opened	'Already opened'

close

Closes the Sidebar UI.

Example

```
oMyPlugin.command('SideBar.close');
```

Resolutions

Status	When	Returns
resolved	When sidebar is successfully closed	n/a
rejected	When sidebar is already closed	'already closed'

expand

To show more details about the channels, Sidebar slides out from the sides of the screen on desktop machines but expands to full screen in mobile devices.

Example

```
oMyPlugin.command('SideBar.expand');
```

Resolutions

Status	When	Returns
resolved	When sidebar is successfully expanded	n/a
rejected	When sidebar is already expanded	'sidebar already expanded'

contract

Retracts the expanded version of Sidebar, showing only the channel buttons on desktop machines and the sidebar launcher button on mobile devices.

Example

```
oMyPlugin.command('SideBar.contract');
```

Resolutions

Status	When	Returns
resolved	When sidebar is successfully contracted	n/a
rejected	When sidebar is already contracted	sidebar already contracted

WebChat

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Customization](#)
 - [1.3 Namespace](#)
 - [1.4 Mobile support](#)
 - [1.5 Screenshots](#)
- [2 Configuration](#)
 - [2.1 Example](#)
 - [2.2 Options](#)
- [3 Localization](#)
 - [3.1 Special values for localization](#)
 - [3.2 Error handling](#)
 - [3.3 Usage](#)
 - [3.4 Inactivity messages](#)
 - [3.5 Default i18n JSON](#)
- [4 API commands](#)
 - [4.1 configure](#)
 - [4.2 open](#)
 - [4.3 close](#)
 - [4.4 minimize](#)
 - [4.5 endChat](#)
 - [4.6 invite](#)
 - [4.7 reInvite](#)
 - [4.8 injectMessage](#)
 - [4.9 showChatButton](#)
 - [4.10 hideChatButton](#)
 - [4.11 showOverlay](#)
 - [4.12 hideOverlay](#)

- [5 API events](#)
- [6 Metadata](#)
 - [6.1 Interaction Lifecycle](#)
 - [6.2 Lifecycle scenarios](#)
 - [6.3 Metadata](#)
- [7 Customizable chat registration form](#)
 - [7.1 Default example](#)
 - [7.2 Properties](#)
 - [7.3 Labels](#)
 - [7.4 Wrappers](#)
 - [7.5 Validation](#)
 - [7.6 Form submit](#)
- [8 Customizable emoji menu](#)
 - [8.1 Introduction](#)
 - [8.2 Differences between v1 and v2](#)
 - [8.3 Configuring the emoji menu](#)
 - [8.4 Localization](#)
- [9 Terminate Chat session on contact side](#)

- Developer

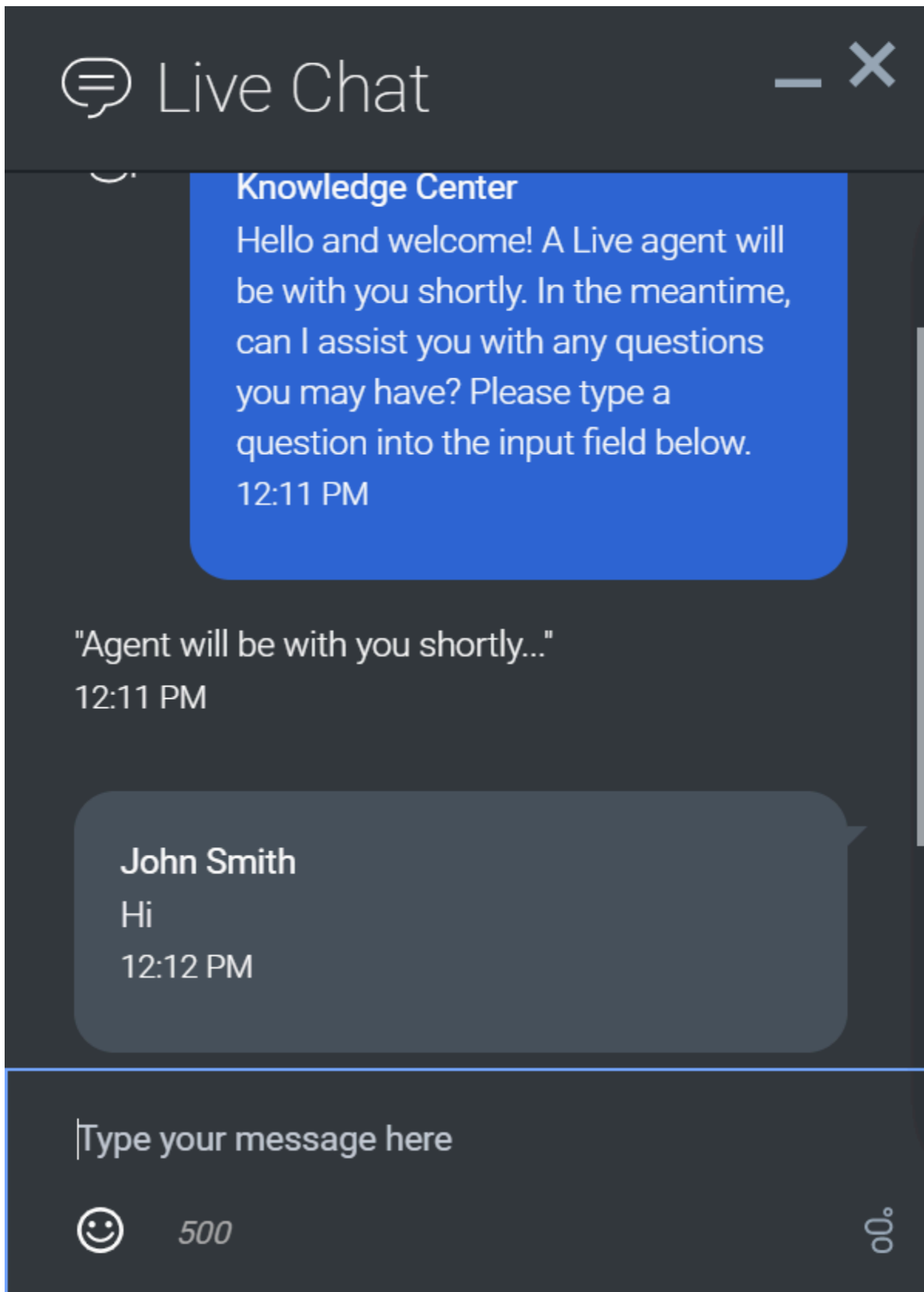
Learn how to enable live chats between customers and agents.

Related documentation:

-

[Link to video](#)

Overview



The WebChat Widget allows a customer to start a live chat with a customer service agent. The UI appears within the page and follows the customer as they traverse your website. Other features include minimize/maximize, auto-reconnect, and a built-in invite feature.

Usage

You can launch WebChat manually by using the following methods:

- Call the **WebChat.open** command
- Configure ChannelSelector to show WebChat as a channel
- Enable the built-in launcher button for WebChat that appears on the right side of the screen
- Create your own custom button or link to open WebChat (using the **WebChat.open** command)

Customization

You can customize and localize all of the static text shown in the WebChat Widget by adding entries to your configuration and localization options.

WebChat supports themes. You can create and register your own themes for Genesys Widgets.

Namespace

The WebChat plugin has the following namespaces tied to each of the following types:

Type	Namespace
Configuration	webchat
i18n—Localization	webchat
CXBus—API commands & API events	WebChat
CSS	.cx-webchat

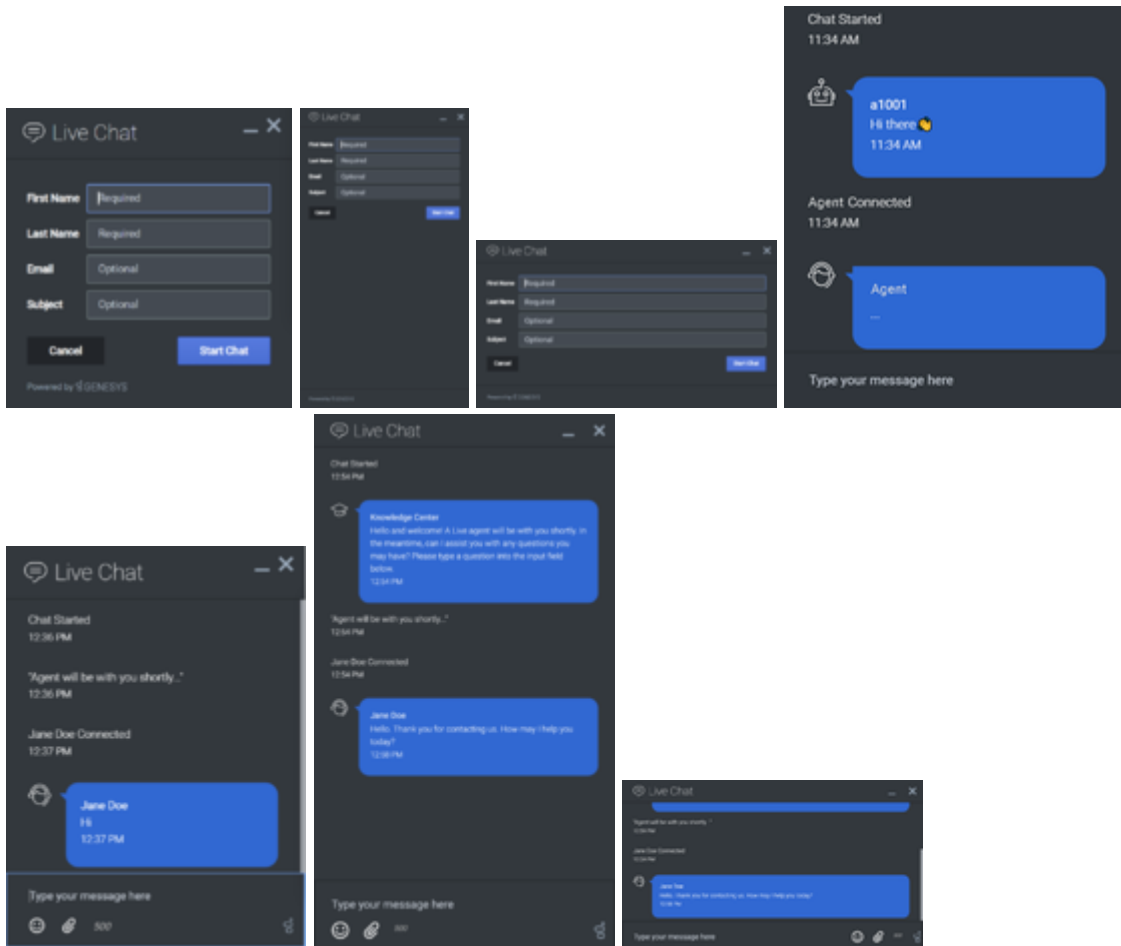
Mobile support

WebChat supports both desktop and mobile devices. Like all Genesys Widgets, there are two main modes: Desktop & Mobile. Desktop is employed for monitors, laptops, and tablets. Mobile is employed for smartphones. When a smartphone is detected, WebChat switches to special full-screen templates that are optimized for both portrait and landscape orientations.

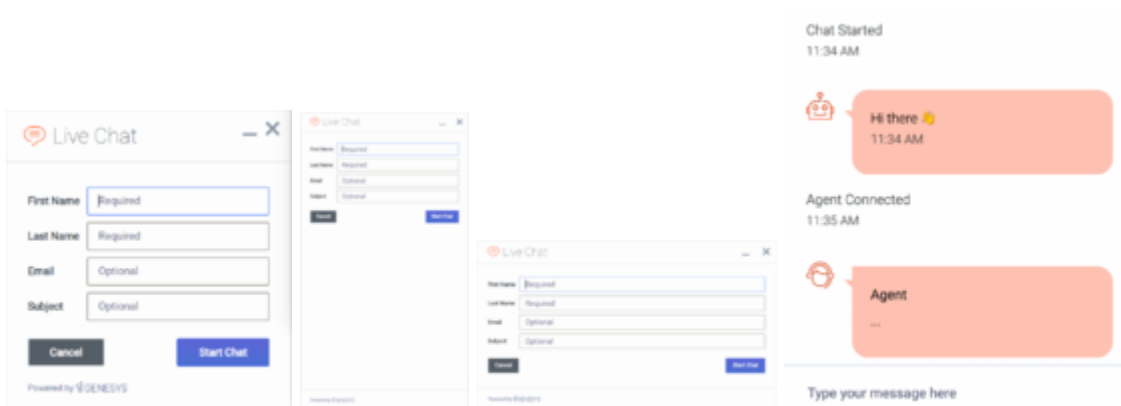
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

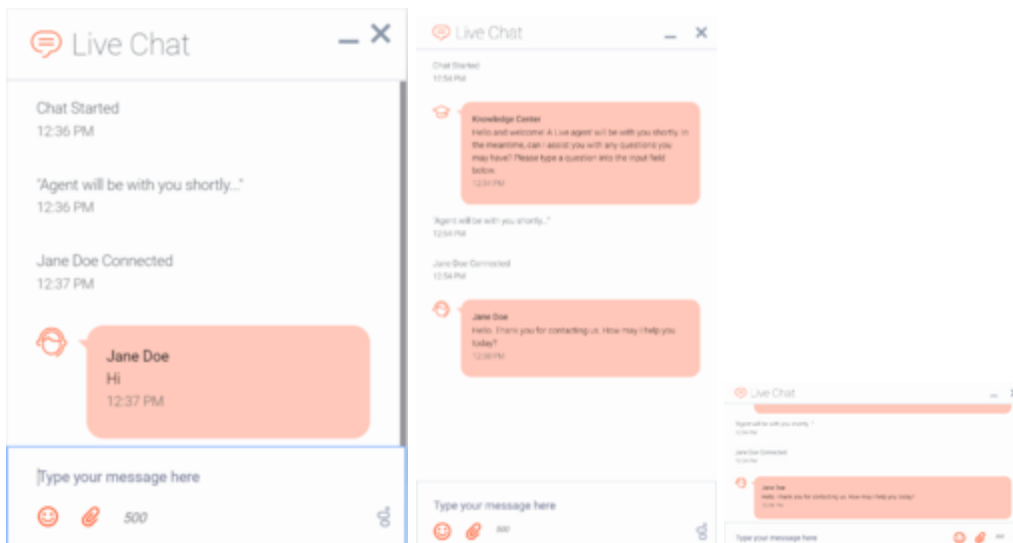
Screenshots

Dark theme



Light theme





Important

The dark theme is active by default. You may also change colors/themes for widgets by following the instructions on the [Customize appearance](#) page.

Configuration

[Link to video](#)

WebChat and WebChatService share the **_genesys.widgets.webchat** configuration namespace. WebChat has UI options while WebChatService has connection options.

Example

```
window._genesys.widgets.webchat = {  
  apiKey: 'n3eNkgLLgLKXREBYjGm6lygOHH0K8VA',  
  dataURL: 'https://api.genesyscloud.com/gms-chat/2/chat',  
  userData: {},  
  emojis: true,  
  uploadsEnabled: false,  
  confirmFormCloseEnabled: true,  
  actionsMenu: true,  
  maxMessageLength: 140,  
  
  autoInvite: {  
    enabled: false,  

```

```
        timeToInviteSeconds: 10,  
        inviteTimeoutSeconds: 30  
    },  
    chatButton: {  
        enabled: true,  
        template: '  
,  
        effect: 'fade',  
        openDelay: 1000,  
        effectDuration: 300,  
        hideDuringInvite: true  
    },  
    minimizeOnMobileRestore: false,  
    ariaIdleAlertIntervals:[50,25,10],  
    ariaCharRemainingIntervals:[75, 25, 10]  
};
```

Options

Name	Type	Description	Default	Required	Introduced/ Updated
emojis	boolean	Enable/disable Emoji menu inside chat message input. Emojis are supported using unicode characters and the list includes 😊 U+263A (smile), 👍 U+1F44D (thumbs up) and 😞 U+2639 (sad).	false	n/a	
form	object	A JSON object containing a custom registration form definition. The JSON definition placed here becomes the default registration form layout for WebChat. See Customizable Chat	A basic registration form is defined internally by default	n/a	

Name	Type	Description	Default	Required	Introduced/ Updated
		Registration Form.			
uploadsEnabled	boolean	Show/Hide the Send File button. The button will be shown if the value is set to true.	false	n/a	
confirmFormCloseEnabled	boolean	Enable or disable displaying a confirmation message before closing WebChat if information has been entered into the registration form.	true	n/a	
timeFormat	number/string	This sets the time format for the timestamps in this widget. It can be 12 or 24.	12	false	
actionsMenu	boolean	Enable/disable actions menu next to chat message input.	true	n/a	
maxMessageLength	number	Set a character limit that the user can input into the message area during a chat. When max is reached, user cannot type any more.	500	n/a	
charCountEnabled	boolean	Show/Hide the number of characters remaining in the input message area while the user is typing.	false	n/a	
autoInvite.enabled	boolean	Enable/disable	false	n/a	

Name	Type	Description	Default	Required	Introduced/ Updated
		<p>auto-invite feature. Automatically invites user to chat after user idles on page for preset time.</p> <div> Important When running Widgets in lazy load mode, this option requires that you pre-load the WebChat plugin. </div>			
autoInvite.timeToInvite	Integer	Number of seconds of idle time before inviting customer to chat.	5	n/a	
autoInvite.inviteTimeout	Integer	Number of seconds to wait, after showing invite, before closing chat invite.	30	n/a	
chatButton.enabled	Boolean	<p>Enable/disable chat button on screen.</p> <div> Important When running Widgets in lazy load mode, this option requires that you pre-load the WebChat plugin. </div>	false	n/a	
chatButton.template	String	Custom HTML string template for chat button.		n/a	
chatButton.effect	String	Type of animation effect when revealing chat button. 'slide' or 'fade'.	fade	n/a	
chatButton.openDelay	Integer	Number of milliseconds before displaying chat	1000	n/a	

Name	Type	Description	Default	Required	Introduced/ Updated
		button on screen.			
chatButton.effectDuration	number	Length of animation effect in milliseconds.	300	n/a	
chatButton.hideOnInvite	boolean	When the auto-invite feature is activated, hides the chat button. When invite is dismissed, reveals the chat button again.	true	n/a	
ariaIdleAlertIntervals	array/boolean	An array containing the intervals as a percentage at which the screen reader will announce the remaining idle time. By default, it is enabled with the following time intervals, and it is customizable according to the user's needs. Configuring a value of 'false' will let the screen reader call out idle time for every change.	[100, 75, 50, 25, 10]	n/a	9.0.016.11
ariaCharRemainingIntervals	array/boolean	An array containing the intervals as a percentage at which the screen reader will announce the remaining characters when the user inputs text into the message	[50, 25, 10]	n/a	9.0.016.11

Name	Type	Description	Default	Required	Introduced/ Updated
		area. By default, it is enabled with the following intervals, and it is customizable according to the user needs. Configuring a value of 'false' will let the screen reader call out remaining characters for every change.			

Localization

You can define string key names and values to match the system messages that are received from the chat server. If a customer system message is received as **SYS001** in the message body, Webchat checks to determine if any keys match in the language pack, and then replaces the message body accordingly. **SYS001** is an example format. There are no format restrictions on custom message keys. The purpose of this feature is to allow localization for the User Interface and Server to be kept in the same file.

Special values for localization

You can inject the special value. When used, the agent's name is rendered in its place at runtime.

Error handling

Customers can define their own error messages in the **Errors** section found in the above Webchat Localization. If no error messages are defined, default error messages are used.

Important

For information on how to set up localization, refer to Localize widgets and services.

Usage

You must use the *webchat* namespace for defining localization strings for the WebChat plugin in your i18n JSON file.

The following example shows how to define new strings for the **en** (English) language. You can use

any language codes you wish; there is no standard format. When selecting the active language in your configuration, you must match one of the language codes defined in your i18n JSON file. Please note that you must only define a language code once in your i18n JSON file. Inside each language object you should define new strings for each widget.

Inactivity messages

If Chat Server is configured to end the chat session after a certain idle time, it may send several warning messages to the client to inform them and prompt them to act. Chat Server can be configured to show a first warning, a second warning, and a final notice when it ends the chat session. By default, WebChat will display the warning message text as it is received from the server. If you wish to localize these methods on the client side instead, follow these steps:

The first warning can be localized by setting the string 'IdleMessage1'.

The second warning can be localized by setting the string 'IdleMessage2'.

The final notice can be localized by setting the string 'IdleMessageClose'.

Tip

If Chat Server ever allows more than two idle warning messages, you can localize them by incrementing the integer value in the string name (e.g. 'IdleMessage3', 'IdleMessage4', and so on).

Default i18n JSON

```
{
  "en": {
    "webchat": {
      "ChatButton": "Chat",
      "ChatStarted": "Chat Started",
      "ChatEnded": "Chat Ended",
      "AgentNameDefault": "Agent",
      "AgentConnected": " Connected",
      "AgentDisconnected": " Disconnected",
      "BotNameDefault": "Bot",
      "BotConnected": " Connected",
      "BotDisconnected": " Disconnected",
      "SupervisorNameDefault": "Supervisor",
      "SupervisorConnected": " Connected",
      "SupervisorDisconnected": " Disconnected",
      "AgentTyping": "...",
      "AriaAgentTyping": "Agent is typing",
      "AgentUnavailable": "Sorry. There are no agents available. Please try
later.",
      "ChatTitle": "Live Chat",
      "ChatEnd": "X",
      "ChatClose": "X",
      "ChatMinimize": "Min",
      "ChatFormFirstName": "First Name",
      "ChatFormLastName": "Last Name",
      "ChatFormNickname": "Nickname",
      "ChatFormEmail": "Email",
```



```
"ChatFormSubject": "Subject",
"ChatFormPlaceholderFirstName": "Required",
"ChatFormPlaceholderLastName": "Required",
"ChatFormPlaceholderNickname": "Optional",
"ChatFormPlaceholderEmail": "Optional",
"ChatFormPlaceholderSubject": "Optional",
"ChatFormSubmit": "Start Chat",
"AriaChatFormSubmit": "Start Chat",
"ChatFormCancel": "Cancel",
"AriaChatFormCancel": "Cancel",
"ChatFormClose": "Close",
"ChatInputPlaceholder": "Type your message here",
"ChatInputSend": "Send",
"AriaChatInputSend": "Send",
"ChatEndQuestion": "Are you sure you want to end this chat session?",
"ChatEndCancel": "Cancel",
"ChatEndConfirm": "End chat",
"AriaChatEndCancel": "Cancel",
"AriaChatEndConfirm": "End chat",
"ConfirmCloseWindow": "Are you sure you want to close chat?",
"ConfirmCloseCancel": "Cancel",
"ConfirmCloseConfirm": "Close",
"AriaConfirmCloseCancel": "Cancel",
"AriaConfirmCloseConfirm": "Close",
"ActionsDownload": "Download transcript",
"ActionsEmail": "Email transcript",
"ActionsPrint": "Print transcript",
"ActionsSendFile": "Attach Files",
"AriaActionsSendFileTitle": "Opens a file upload dialog",
"ActionsEmoji": "Send Emoji",
"ActionsVideo": "Invite to Video Chat",
"ActionsTransfer": "Transfer",
"ActionsInvite": "Invite",
"InstructionsTransfer": "Open this link on another device to transfer
your chat session",
"InstructionsInvite": "Share this link with another person to add
them to this chat session",
"InviteTitle": "Need help?",
"InviteBody": "Let us know if we can help out.",
"InviteReject": "No thanks",
"InviteAccept": "Start chat",
"AriaInviteAccept": "Start chat",
"AriaInviteReject": "No thanks",
"ChatError": "There was a problem starting the chat session. Please
retry.",
"ChatErrorButton": "OK",
"AriaChatErrorButton": "OK",
"ChatErrorPrimaryButton": "Yes",
"ChatErrorDefaultButton": "No",
"AriaChatErrorPrimaryButton": "Yes",
"AriaChatErrorDefaultButton": "No",
"DownloadButton": "Download",
"AriaDownloadButton": "Download",
"FileSent": "has sent:",
"FileTransferRetry": "Retry",
"AriaFileTransferRetry": "Retry",
"FileTransferError": "OK",
"AriaFileTransferError": "OK",
"FileTransferCancel": "Cancel",
"AriaFileTransferCancel": "Cancel",
"RestoreTimeoutTitle": "Chat ended",
"RestoreTimeoutBody": "Your previous chat session has timed out.
Would you like to start a new one?",
```

```

"RestoreTimeoutReject": "No thanks",
"RestoreTimeoutAccept": "Start chat",
"AriaRestoreTimeoutAccept": "Start chat",
"AriaRestoreTimeoutReject": "No thanks",
"EndConfirmBody": "Would you really like to end your chat session?",
"EndConfirmAccept": "End chat",
"EndConfirmReject": "Cancel",
"AriaEndConfirmAccept": "End chat",
"AriaEndConfirmReject": "Cancel",
"SurveyOfferQuestion": "Would you like to participate in a survey?",
"ShowSurveyAccept": "Yes",
"ShowSurveyReject": "No",
"AriaShowSurveyAccept": "Yes",
"AriaShowSurveyReject": "No",
"UnreadMessagesTitle": "unread",
"AriaYouSaid": "You said",
"AriaSaid": "said",
"AriaSystemSaid": "System said",
"AriaWindowLabel": "Live Chat Window",
"AriaMinimize": "Live Chat Minimize",
"AriaMaximize": "Live Chat Maximize",
"AriaClose": "Live Chat Close",
"AriaEmojiStatusOpen": "Emoji picker dialog is opened",
"AriaEmojiStatusClose": "Emoji picker dialog is closed",
"AriaEmoji": "emoji",
"AriaCharRemaining": "Characters remaining",
"AriaMessageInput": "Message box",
"AsyncChatEnd": "End Chat",
"AsyncChatClose": "Close Window",
"AriaAsyncChatEnd": "End Chat",
"AriaAsyncChatClose": "Close Window",
"DayLabels": [
    "Sun",
    "Mon",
    "Tue",
    "Wed",
    "Thur",
    "Fri",
    "Sat"
],
"MonthLabels": [
    "Jan",
    "Feb",
    "Mar",
    "Apr",
    "May",
    "Jun",
    "Jul",
    "Aug",
    "Sept",
    "Oct",
    "Nov",
    "Dec"
],
"todayLabel": "Today",
"errors": {
    "102": "First name is required.",
    "103": "Last name is required.",
    "161": "Please enter your name.",
    "201": "The file could not be sent."
}

```

..

The maximum number of attached files would be exceeded ().

```
" ,
                                "202": "The file could not be sent.
"
Upload limit and/or maximum number of attachments would be exceeded ().
" ,
                                "203": "The file could not be sent.
"
File type is not allowed.
" ,
                                "204": "We're sorry but your message is too long. Please
write a shorter message.",
                                "240": "We're sorry but we cannot start a new chat at this
time. Please try again later.",
                                "364": "Invalid email address.",
                                "401": "We're sorry but we are not able to authorize the chat
session. Would you like to start a new chat?",
                                "404": "We're sorry but we cannot find your previous chat
session. Would you like to start a new chat?",
                                "500": "We're sorry, an unexpected error occurred with the
service. Would you like to close and start a new Chat?",
                                "503": "We're sorry, the service is currently unavailable or
busy. Would you like to close and start a new Chat again?",
                                "ChatUnavailable": "We're sorry but we cannot start a new
chat at this time. Please try again later.",
                                "CriticalFault": "Your chat session has ended unexpectedly
due to an unknown issue. We apologize for the inconvenience.",
                                "StartFailed": "There was an issue starting your chat
session. Please verify your connection and that you submitted all required information
properly, then try again.",
                                "MessageFailed": "Your message was not received successfully.
Please try again.",
                                "RestoreFailed": "We're sorry but we were unable to restore
your chat session due to an unknown error.",
                                "TransferFailed": "Unable to transfer chat at this time.
Please try again later.",
                                "FileTransferSizeError": "The file could not be sent.
"
File size is larger than the allowed size ().
" ,
                                "InviteFailed": "Unable to generate invite at this time.
Please try again later.",
                                "ChatServerWentOffline": "Messages are currently taking
longer than normal to get through. We're sorry for the delay.",
                                "RestoredOffline": "Messages are currently taking longer than
normal to get through. We're sorry for the delay.",
                                "Disconnected": "
Connection lost
" ,
                                "Reconnected": "
Connection restored
" ,
                                "FileSendFailed": "The file could not be sent.
There was an unexpected disconnection. Try again?
" ,
                                "Generic": "
An unexpected error occurred.
```

```

",
    "pureengage-v3-rest-INVALID_FILE_TYPE": "Invalid file type.
Only Images are allowed.",
    "pureengage-v3-rest-LIMIT_FILE_SIZE": "File size is larger
than the allowed size.",
    "pureengage-v3-rest-LIMIT_FILE_COUNT": "The maximum number of
attached files exceeded the limit.",
    "pureengage-v3-rest-INVALID_CONTACT_CENTER": "Invalid x-api-
key transport configuration.",
    "pureengage-v3-rest-INVALID_ENDPOINT": "Invalid endpoint
transport configuration.",
    "pureengage-v3-rest-INVALID_NICKNAME": "First Name is
required.",
    "pureengage-v3-rest-AUTHENTICATION_REQUIRED": "We're sorry
but we are not able to authorize the chat session.",
    "purecloud-v2-sockets-400": "Sorry, something went wrong.
Please verify your connection and that you submitted all required information properly, then
try again.",
    "purecloud-v2-sockets-500": "We're are sorry, an unexpected
error occurred with the service.",
    "purecloud-v2-sockets-503": "We're sorry, the service is
currently unavailable."
  }
}
}
}

```

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see [Genesys Widgets Extensions](#) for more information about extending Genesys Widgets.

```

var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');
oMyPlugin.command('WebChat.open');

```

configure

Internal use only. The main App plugin shares configuration settings to widgets using each widget's `configure` command. The `configure` command can only be called once at startup. Calling `configure` again after startup may result in unpredictable behavior.

open

Opens the WebChat UI.

Example

```
oMyPlugin.command('WebChat.open', {
    userData: {},
    form: {
        autoSubmit: false,
        firstname: 'John',
        lastname: 'Smith',
        email: 'John@mail.com',
        subject: 'Customer Satisfaction'
    }
    formJSON: {...}
    markdown: false
}).done(function(e){
    // WebChat opened successfully
}).fail(function(e){
    // WebChat isn't open or no active chat session
});
```

Options

Option	Type	Description
form	object	Object containing form data to prefill in the chat entry form and optionally auto-submit the form.
form.autoSubmit	boolean	Automatically submit the form. Useful for bypassing the entry form step.
form.firstname	string	Value for the first name entry field.
form.lastname	string	Value for the last name entry field.
form.email	string	Value for the email entry field.
form.subject	string	Value for the subject entry field.
formJSON	object	An object containing a custom registration form definition. See Customizable chat registration form.
userData	object	Object containing arbitrary data that gets sent to the server. Overrides userData set in the webchat configuration object.
async	boolean	Starts a new chat either in

Option	Type	Description
		asynchronous or normal mode based on the boolean value. Note that unless async static configuration is defined, a chat in normal mode will start automatically.
markdown	boolean	The markdown feature for chat messages.
id	string	A Unique identifier of a chat session that helps to identify the instance of that session and its associated events. A random value is automatically generated and assigned when no value is passed explicitly.

Resolutions

Status	When	Returns
resolved	WebChat is successfully opened	n/a
rejected	WebChat is already open	'already opened'

close

Closes the WebChat UI.

Example

```
oMyPlugin.command('WebChat.close').done(function(e){
    // WebChat closed successfully
}).fail(function(e){
    // WebChat is already closed or no active chat session
});
```

Resolutions

Status	When	Returns
resolved	WebChat is successfully closed	n/a
rejected	WebChat is already closed	'already closed'

minimize

Minimizes or un-minimizes the WebChat UI.

Example

```
oMyPlugin.command('WebChat.minimize').done(function(e){  
    // WebChat minimized successfully  
}).fail(function(e){  
    // WebChat ignores command  
});
```

Options

Option	Type	Description
minimized	boolean	Rather than toggling the current minimized state you can specify the minified state directly: true = minimized, false = unminimized.

Resolutions

Status	When	Returns
resolved	Always	n/a
rejected	Never	'Invalid configuration'

endChat

Starts the **end chat** procedure. User may be prompted to confirm.

Example

```
oMyPlugin.command('WebChat.endChat').done(function(e){  
    // WebChat ended a chat successfully  
}).fail(function(e){  
    // WebChat has no active chat session  
});
```

Resolutions

Status	When	Returns
resolved	There is an active chat session to end	n/a
rejected	There is no active chat session to end	'there is no active chat session to end'

invite

Shows an invitation to chat using the Toaster popup element. The text shown in the invitation can be edited in the localization file.

Example

```
oMyPlugin.command('WebChat.invite').done(function(e){  
    // WebChat invited successfully  
}).fail(function(e){  
    // WebChat is already open and will be ignored  
});
```

Resolutions

Status	When	Returns
resolved	WebChat is closed and the toast element is created successfully	n/a
rejected	WebChat is already open (prevents inviting a user that is already in a chat)	'Chat is already open. Ignoring invite command.'

reInvite

When an active chat session cannot be restored, this invitation offers to start a new chat for the user. The text shown in the invitation can be edited in the localization file.

Example

```
oMyPlugin.command('WebChat.reInvite').done(function(e){  
    // WebChat reinvited successfully  
}).fail(function(e){  
    // WebChat is already open and will be ignored  
});
```

Resolutions

Status	When	Returns
resolved	WebChat is closed, the config item 'webchat.inviteOnRestoreTimeout' is set, and the toast element is created successfully	n/a
rejected	WebChat is already open (prevents inviting a user that is already in a chat)	'Chat is already open. Ignoring invite command.'

injectMessage

Injects a custom message into the chat transcript. Useful for extending WebChat functionality with other Genesys products.

Example

```
oMyPlugin.command('WebChat.injectMessage', {  
  type: 'text',  
  name: 'person',  
  text: 'hello',  
  custom: false,  
  bubble:{  
    fill: '#00FF00',  
    radius: '4px',  
    time: false,  
    name: false,  
    direction: 'right',  
    avatar:{  
      custom: '  
word  
'',  
      icon: 'email'  
    }  
  }  
}).done(function(e){  
  // WebChat injected a message successfully  
  // e.data == The message HTML reference (jQuery wrapped set)  
}).fail(function(e){  
  // WebChat isn't open or no active chat  
});
```

Options

Option	Type	Description
type	string	Switch the rendering type of the injected message between text and html.
name	string	Specify a name label for the message to identify what service or widget has injected the message.
text	string	The content of the message. Either plain text or HTML.
custom	boolean	If set to true, the default message template will not be used, allowing you to inject a highly customized HTML block unconstrained by the normal message template.
bubble.fill	string of valid CSS color value	The content of the message. Either plain text or HTML.
bubble.radius	string of valid CSS border radius value	The border radius you'd like for the bubble.

Option	Type	Description
bubble.time	boolean	If you'd like to show the timestamp for the bubble.
bubble.name	boolean	If you'd like to show the name for the bubble.
bubble.direction	string	Which direction you want the message bubble to come from.
bubble.avatar.custom	string or HTML reference	Change the content of the html that would be the avatar for the chat bubble.
bubble.avatar.icon	class name	Generated common library provided for icon name.

Resolutions

Status	When	Returns
resolved	WebChat is open and there is an active chat session	An HTML reference (jQuery wrapped set) to the new injected message.
rejected	WebChat is not open and/or there was no active chat session	'No chat session to inject into'

showChatButton

Displays the standalone chat button using either the default template and CSS, or customer-defined ones.

Example

```
oMyPlugin.command('WebChat.showChatButton', {  
    openDelay: 1000,  
    duration: 1500  
}).done(function(e){  
    // WebChat shows chat button successfully  
}).fail(function(e){  
    // WebChat button is already visible, side bar is active and overrides the chat  
    button, or chat button is disabled in configuration  
});
```

Options

Option	Type	Description
openDelay	number	Duration in milliseconds to delay showing the chat button on the page.
duration	number	Duration in milliseconds for the

Option	Type	Description
		show and hide animation.

Resolutions

Status	When	Returns
resolved	The chat button is enabled in the configuration, is currently not visible, and the SideBar plugin is not initialized	n/a
rejected	The chat button is not enabled in the configuration, or it's already visible, or the SideBar plugin is initialized	'Chat button is already visible. Ignoring command.'
rejected	The SideBar plugin is active the standalone chat button will be disabled automatically	'SideBar is active and overrides the default chat button'

hideChatButton

Hides the standalone chat button.

Example

```
oMyPlugin.command('WebChat.hideChatButton', {duration: 1500}).done(function(e){  
    // WebChat hid chat button successfully  
}).fail(function(e){  
    // WebChat button is already hidden  
});
```

Options

Option	Type	Description
duration	number	Duration in milliseconds for the show and hide animation.

Resolutions

Status	When	Returns
resolved	The chat button is currently visible	n/a
rejected	The chat button is already hidden	'Chat button is already hidden. Ignoring command.'

showOverlay

Opens a slide-down overlay over WebChat's content. You can fill this overlay with content such as

disclaimers, articles, and other information.

Example

```
oMyPlugin.command('WebChat.showOverlay', {
    html: '
Example text
',
    hideFooter: false
}).done(function(e){
    // WebChat successfully shows overlay
}).fail(function(e){
    // WebChat isn't open
});
```

Options

Option	Type	Description
html	string or HTML reference	The HTML content you want to display in the overlay.
hideFooter	boolean	Normally the overlay appears between the titlebar and footer bar. Set this to true to have the overlay overlap the footer to gain a bit more vertical space. This should only be used in special cases. For general use, don't set this value.

Resolutions

Status	When	Returns
resolved	WebChat is open and the overlay opens	
rejected	WebChat is not currently open	WebChat is not currently open. Ignoring command.

hideOverlay

Hides the slide-down overlay.

Example

```
oMyPlugin.command('WebChat.hideOverlay').done(function(e){
    // WebChat hid overlay successfully
}).fail(function(e){
```

```
    // WebChat isn't open  
});
```

Resolutions

Status	When	Returns
resolved	WebChat is open and the overlay closes	
rejected	WebChat is not currently open	WebChat is not currently open. Ignoring command.

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets Extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('WebChat.ready', function(e){});
```

Name	Description	Data
ready	WebChat is initialized and ready to accept commands	n/a
opened	The WebChat widget has appeared on screen	n/a
started	The WebChat has successfully started.	Metadata
submitted	The user has submitted the form.	Metadata
rejected	When the chat session fails to start. Typically due to form validation or network errors.	Metadata
completed	The Chat session ended after agent is successfully connected to WebChat.	Metadata
cancelled	The Chat session ended before agent is connected to WebChat.	Metadata
closed	The WebChat widget has been removed from the screen	Metadata
minimized	The WebChat widget has been	n/a

Name	Description	Data
	changed to a minimized state	
unminimized	The WebChat widget has been restored from a minimized state to the standard view	n/a
messageAdded	When a message is added to the transcript, this event will fire	Returns an object containing two properties: 'data' and 'html'. 'data' contains the JSON data for the message, while 'html' contains a reference to the visible message inside the chat transcript.

Metadata

Interaction Lifecycle

Every WebChat interaction has a sequence of events we call the *Interaction Lifecycle*. This is a sequence of events that tracks progress and choices from the beginning of an interaction (opening WebChat), to the end (closing WebChat), and every step in between.

The following events are part of the Interaction Lifecycle:

ready
opened
started
cancelled
submitted
rejected
completed
closed

Lifecycle scenarios

An Interaction Lifecycle can vary based on each user's intent and experience with WebChat. Here are several sequences of events in the lifecycle that correspond to different scenarios.

The user opened WebChat but changed their mind and closed it without starting a chat session:

ready -> opened -> cancelled -> closed

The user started a chat session but ended it before an agent connected. Perhaps it was taking too long to reach someone:

ready -> opened -> started -> cancelled -> closed

The user started a chat, but the chat fails to start:

ready -> opened -> started -> submitted -> rejected

The user started a chat, met with an agent, and the session ended normally:

ready -> opened -> started -> submitted -> completed -> closed

Tip

For a list of all WebChat events, see API events.

Metadata

Each event in the Interaction Lifecycle includes the following block of metadata. By default, all values are set to false. As the user progresses through the lifecycle of a WebChat interaction, these values will be updated.

The metadata block contains boolean state flags, counters, timestamps, and elapsed times. These values can be used to track and identify trends or issues with chat interactions. During run-time, the metadata can help you offer a smart and dynamic experience to your users.

Reference

Name	Type	Description
proactive	boolean	Indicates this chat session was started proactively.
prefilled	boolean	Indicates the registration form was prefilled with info automatically.
autoSubmitted	boolean	Indicates the registration form was submitted automatically, usually after being prefilled.
filesUploaded	integer	Current number of files uploaded during chat session.
numAgents	integer	Current number of agents that have connected to the chat session.
userMessages	integer	Current number of messages sent by user.
agentMessages	integer	Current number of messages sent by agents.
systemMessages	integer	Current number of system messages received.
errors	array/boolean	An array of error codes encountered during chat session. If no errors, this value will be false.
form	object	An object containing the form parameters when the form is submitted.
opened	integer (timestamp)	Timestamp indicating when WebChat was opened.

Name	Type	Description
started	integer (timestamp)	Timestamp indicating when chat session started.
cancelled	integer (timestamp)	Timestamp indicating when the chat session was cancelled. Cancelled refers to when a user ends a chat session before an agent connects.
rejected	integer (timestamp)	Timestamp indicating when the chat session was rejected. Rejected refers to when a chat session fails to start.
completed	integer (timestamp)	Timestamp indicating when the chat session ended normally. Completed refers to when a user or agent ends a chat after an agent connected.
closed	integer (timestamp)	Timestamp indicating when WebChat was closed.
agentReached	integer (timestamp)	Timestamp indicating when the first agent was reached, if any.
supervisorReached	integer (timestamp)	Timestamp indicating when the first agent supervisor was reached, if any.
elapsed	integer (milliseconds)	Total elapsed time in milliseconds from when the user started the chat session to when the chat session ended.
waitingForAgent	integer (milliseconds)	Total time in milliseconds waiting for an agent from when the user started the chat session to when an agent connected to the session.
id	string	A Unique identifier of a chat session that helps to identify the instance of that session and its associated events.

Customizable chat registration form

WebChat allows you to customize the registration form shown to users prior to starting a session. The following form inputs are currently supported:

- Text
- Select
- Hidden
- Checkbox

- Textarea

Customization is done through a JSON object structure that defines the layout, input type, label, and attributes for each input. You can set the default registration form definition in the **`_genesys.widgets.webchat.form`** configuration option. Alternately, you can pass a new registration form definition through the `WebChat.open` command:

```
_genesys.widgets.bus.command("WebChat.open", {formJSON: oRegFormDef});
```

Inputs are rendered as stacked rows with one input and one optional label per row.

Default example

The following example is the default JSON object used to render WebChat's registration form. This is a very simple definition that does not use many properties.

```
{
  wrapper: "
", inputs: [ { id: "cx_webchat_form_firstname", name: "firstname", maxlength:
"100", placeholder: "@i18n:webchat.ChatFormPlaceholderFirstName", label:
"@i18n:webchat.ChatFormFirstName" }, { id: "cx_webchat_form_lastname",
name: "lastname", maxlength: "100", placeholder:
"@i18n:webchat.ChatFormPlaceholderLastName", label:
"@i18n:webchat.ChatFormLastName" }, { id: "cx_webchat_form_email", name:
"email", maxlength: "100", placeholder:
"@i18n:webchat.ChatFormPlaceholderEmail", label:
"@i18n:webchat.ChatFormEmail" }, { id: "cx_webchat_form_subject", name:
"subject", maxlength: "100", placeholder:
"@i18n:webchat.ChatFormPlaceholderSubject", label:
"@i18n:webchat.ChatFormSubject" } ] }
```

This JSON definition generates the following output:

The image shows a 'Live Chat' window with a dark theme. At the top, there's a title bar with a chat icon and the text 'Live Chat', and window control buttons (minimize, maximize, close). Below the title bar, there are four input fields arranged vertically. Each field has a label to its left and a status indicator to its right. The first field is 'First Name' with a 'Required' status. The second is 'Last Name' with a 'Required' status. The third is 'Email' with an 'Optional' status. The fourth is 'Subject' with an 'Optional' status. Below these fields are two buttons: a 'Cancel' button on the left and a 'Start Chat' button on the right. At the bottom left, there is a small logo and the text 'Powered by GENESYS'.

Properties

Each input definition can contain any number of properties. These are categorized in two groups: "Special properties", which are custom properties used internally to handle rendering logic, and "HTML attributes" which are properties that are applied directly as HTML attributes on the input element.

Special properties

Property	Type	Default	Description
type	string	"text"	Sets the type of input to render. Possible values are currently "text", "hidden", "select", "checkbox", and "textarea".
label	string		Set the text for the label. If no value provided, no label will be shown. You may use localization query strings to enable custom localization (for

Property	Type	Default	Description
			example, label: "@i18n:namespace.StringName"). Localization query strings allow you to use strings from any widget namespace or to create your own namespace in the localization file (i18n.json) and use strings from there (for example, label: "@i18n:myCustomNamespace.myCustomLabel"). For more information, see the Labels section.
wrapper	HTML string	" "	Each input exists in its own row in the form. By default this is a table-row with the label in the left cell and the input in the right cell. You can redefine this wrapper and layout by specifying a new HTML row structure. See the Wrappers section for more info. The default wrapper for an input is "
validate	function		Define a validation function for the input that executes when the input loses focus (blur) or changes value. Your function must return true or false. True to indicate it passed, false to indicate it failed. If your validation fails, the form will not submit and the invalid input will be highlighted in red. See the Validation section for more details and examples.
validateWhileTyping	boolean	false	Execute validation on keypress in addition to blur and change. This ignores non-character keys like shift, ctrl, and alt.
options	array	[]	When 'type' is set to 'select', you can populate the select by

Property	Type	Default	Description
			adding options to this array. Each option is an object (for example, {text: 'Option 1', value: '1'} for a selectable option, and {text: "Group 1", group: true} for an option group).

HTML attributes

With the exception of special properties, all properties will be added as HTML attributes on the input element. You can use standard HTML attributes or make your own.

Example

```
{
  id: "cx_webchat_form_firstname",
  name: "firstname",
  maxlength: "100",
  placeholder: "@i18n:webchat.ChatFormPlaceholderFirstName",
  label: "@i18n:webchat.ChatFormFirstName"
}
```

In this example, `id`, `name`, `maxlength`, and `placeholder` are all standard HTML attributes for the text input element. Whatever values are set here will be applied to the input as HTML attributes.

Important

The default input type is "text", so `type` does not need to be defined if you intend to make a text input.

HTML output

Disabling autocomplete

Since the custom form feature supports adding any HTML attributes to your inputs, you can control standard HTML features like disabling autocomplete. To disable autocomplete, add **autocomplete: "off"** to your input definition.

Example

```
{
  id: "cx_webchat_form_firstname",
  name: "firstname",
  maxlength: "100",
  placeholder: "@i18n:webchat.ChatFormPlaceholderFirstName",
  label: "@i18n:webchat.ChatFormFirstName",
  autocomplete: "off"
}
```

Labels

A label tag will be generated for your input if you specify label text and if your custom input wrapper includes a '{label}' designation. If you have added an ID attribute for your input, the label will automatically be linked to your input so that clicking on the label selects the input or, for checkboxes, toggles it.

Labels can be defined as static strings or localization queries.

Wrappers

Wrappers are HTML string templates that define a layout. There are two kinds of wrappers, **Form Wrappers** and **Input Wrappers**:

Form wrapper

You can specify the parent wrapper for the overall form in the top-level "wrapper" property. In the example below, we specify this value as "

".

This is the default wrapper for the WebChat form:

```
{
    wrapper: "
", /* form wrapper */ inputs: [] }
```

Input wrapper

Each input is rendered as a table row inside the Form Wrapper. You can change this by defining a new wrapper template for your input row. Inside your template you can specify where you want the input and label to be by adding the identifiers "{label}" and "{input}" to your wrapper value. See the example below:

```
{
    id: "cx_webchat_form_firstname",
    name: "firstname",
    maxlength: "100",
    placeholder: "@i18n:webchat.ChatFormPlaceholderFirstName",
    label: "@i18n:webchat.ChatFormFirstName",
    wrapper: "{label}{input}" /* input row wrapper */
}
```

The {label} identifier is optional. Omitting it will allow the input to fill the row. If you decide to keep the label, you can move it to any location within the wrapper, such as putting the label on the right, or stacking the label on top of the input. You can control the layout of each row independently, depending on your needs.

You are not restricted to using a table for your form. You can change the form wrapper to "

" and then change the individual input wrappers from a table-row to your own

specification. Be aware though that when you move away from the default table wrappers, you are responsible for styling and aligning your layout. Only the default table-row wrapper is supported by default Themes and CSS.

Validation

You can apply a validation function to each input that lets you check the value after a change has been made and/or the user has moved to a different input (on change and on blur). You can enable validation on key press by setting `validateWhileTyping` to `true` in your input definition.

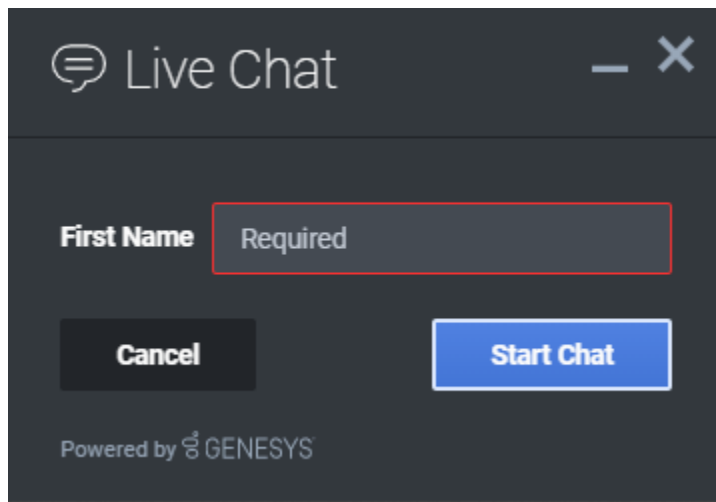
Here is how to define a validation function:

```
{
  id: "cx_webchat_form_firstname",
  name: "firstname",
  maxlength: "100",
  placeholder: "@i18n:webchat.ChatFormPlaceholderFirstName",
  label: "@i18n:webchat.ChatFormFirstName",

  validateWhileTyping: true, // default is false

  validate: function(event, form, input, label, $, CXBus, Common){
    return true; // or false
  }
}
```

You must return `true` or `false` to indicate that validation has passed or failed, respectively. If you return `false`, the WebChat form will not submit, and the input will be highlighted in red. This is achieved by adding the CSS class `"cx-error"` to the input. The image below displays the the field where a user input validation error has occurred, with the field highlighted in red.



Validation function arguments

Argument	Type	Description
event	JavaScript event object	The input event reference object

Argument	Type	Description
		related to the form input field. This event data can be helpful to perform actions like active validation on an input field while the user is typing.
form	HTML reference	A jquery reference to the form wrapper element.
input	HTML reference	A jquery reference to the input element being validated.
label	HTML reference	A jquery reference to the label for the input being validated.
\$	jquery instance	Widget's internal jquery instance. Use this to help you write your validation logic, if needed.
CXBus	CXBus instance	Widget's internal CXBus reference. Use this to call commands on the bus, if needed.
Common	Function Library	Widget's internal Common library of functions and utilities. Use if needed.

Form submit

Custom input field form values are submitted to the server as key value pairs under the `userData` section of the form submit request, where input field names will be the property keys. During the submit, this data is merged along with the `userData` defined in the `WebChat.open` command.

Important

Depending on the API used (PureEnagage V2 API or Genesys Cloud CX) the payload structure in the request can vary for each, but the section below explains how the form data is submitted by the WebChat UI plugin when using custom forms.

Below is the internal form data object defined in the WebChat plugin by default. Since `firstname`, `lastname`, `nickname`, `email`, and `subject` are reserved keywords, users are not allowed to have custom fields with the same name.

```
{
  firstname: '',
  lastname: '',
  nickname: '',
  email: '',
  subject: '',
  userData: {}
}
```

Important

Once the Chat is started, the customer messages display either the nickname or the firstname specified during registration as the **Name**.

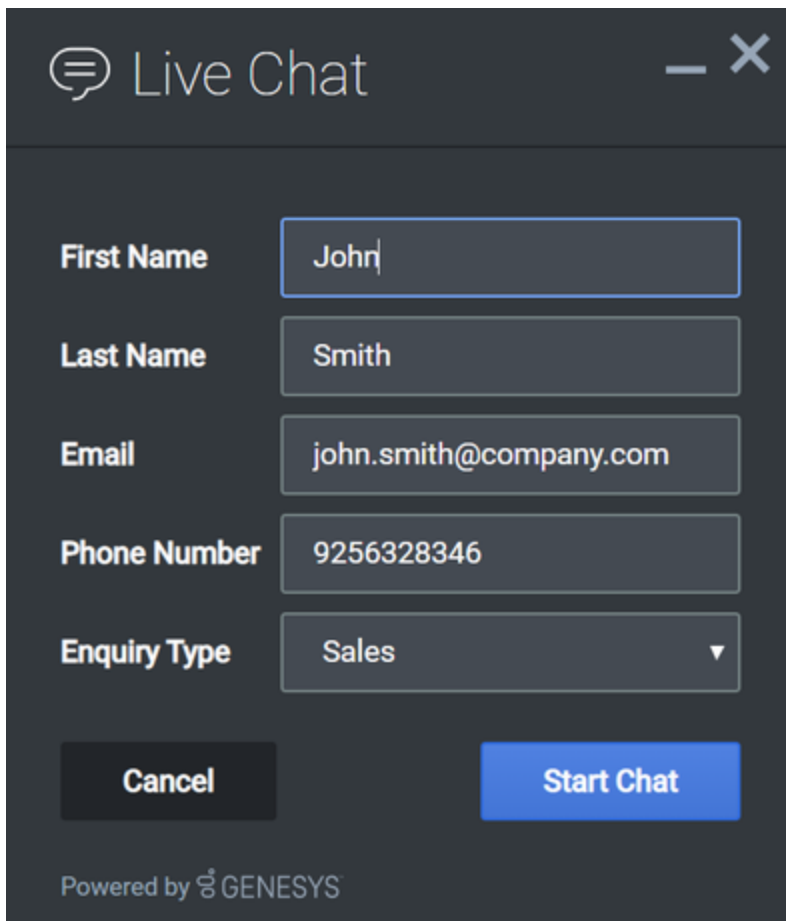
Example

The example below shows how the custom form data given in the WebChat form fields have been mapped as a form data object.

The form fields with reserved keywords like firstname, lastname, and email will be sent as top level and the rest of the fields will be sent under userData to the WebChatService plugin.

Once the form data object is sent to the WebChatService plugin, it will parse and send in the payload request.

```
{
  "wrapper": "
", "inputs": [ { "id": "cx_webchat_form_firstname", "name": "firstname",
"type": "text", "maxlength": "100",
"placeholder": "@i18n:webchat.ChatFormPlaceholderFirstName",
"label": "@i18n:webchat.ChatFormFirstName", "value": "John" }, {
"id": "cx_webchat_form_lastname", "name": "lastname", "type": "text",
"maxlength": "100",
"placeholder": "@i18n:webchat.ChatFormPlaceholderLastName",
"label": "@i18n:webchat.ChatFormLastName", "value": "Smith" }, {
"id": "cx_webchat_form_email", "name": "email", "type": "text", "maxlength": "100",
"placeholder": "@i18n:webchat.ChatFormPlaceholderEmail", "label": "Email",
"value": "john.smith@company.com" }, { "id": "cx_webchat_form_phonenumber",
"name": "phonenumber", "type": "text", "maxlength": "100", "placeholder": "Phone
Number", "label": "Phone Number", "value": "9256328346" }, {
"id": "cx_webchat_form_enquirytype", "name": "enquirytype", "type": "select",
"label": "Enquiry Type", "options": [ { "text": "Account", "group": true }, {
"text": "Sales", "value": "Sales", "selected": true }, { "text": "Credit Card",
"value": "credit card" }, { "text": "General", "group": true }, { "text": "Warranty",
"value": "warranty" }, { "text": "Return policy", "value": "returns" } ] } ] }
```

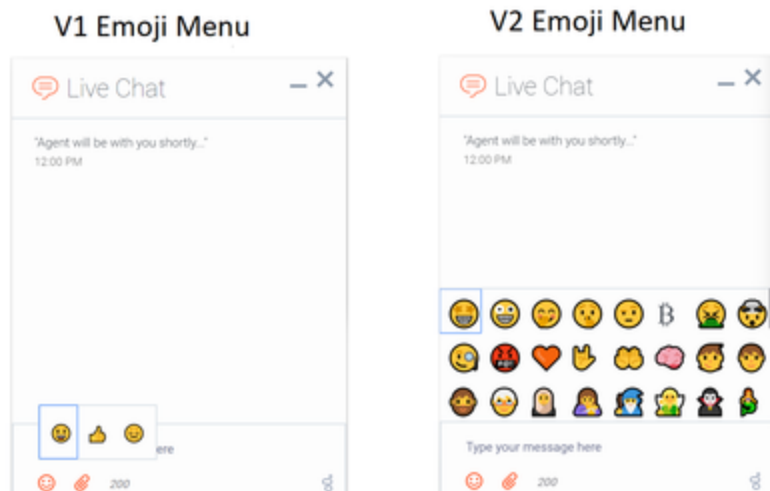

A dark-themed 'Live Chat' window with a title bar containing a speech bubble icon and window controls. The form includes five input fields: 'First Name' (John), 'Last Name' (Smith), 'Email' (john.smith@company.com), 'Phone Number' (9256328346), and 'Enquiry Type' (Sales dropdown). At the bottom are 'Cancel' and 'Start Chat' buttons, and a 'Powered by GENESYS' logo.

```
{
  firstname: 'John',
  lastname: 'Smith',
  email: 'john.smith@company.com',
  userData: {
    phonenumber: '9256328346',
    enquirytype: 'Sales' //value selected from the dropdown
  }
}
```

Customizable emoji menu

Introduction

WebChat offers a v2 emoji menu that lets you choose which emojis to include in the emoji menu.



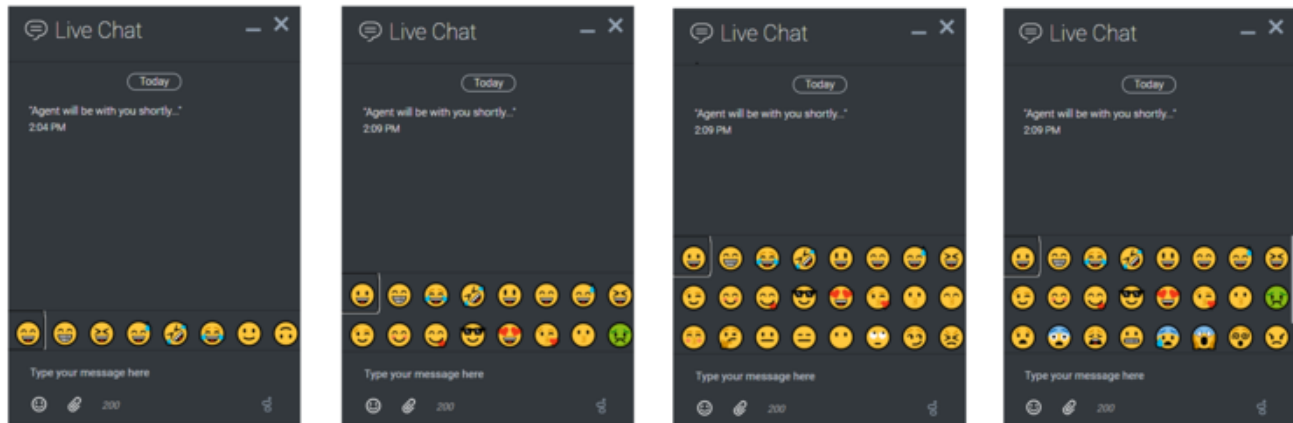
Differences between v1 and v2

- v1 shows as a tooltip-style overlay; v2 shows as a new block between the transcript and the message input.
- v1 closes when you select an emoji or click outside the menu; v2 lets you choose multiple emojis and only closes if you click the emoji menu button again.
- v1 has three fixed emojis to choose from; v2 can show hundreds of customizable emojis in a grid layout.
- v1 menu appears in mobile mode; v2 menu is not available in mobile mode (when v2 is configured, no emoji menu button is present in mobile mode).
- v1 menu has default emojis; v2 menu does not have default emojis. It must be explicitly configured with a list of emojis.

Configuring the emoji menu

Click the emoji menu icon at the bottom-left corner of the WebChat UI to open the v2 emoji menu. The transcript will be resized to fit the emoji menu, which can vary in height depending on the number of emojis configured.

- When 1-8 emojis are configured, the menu has one row, and no scrollbar appears.
- When 9-16 emojis are configured, the menu has two rows, and no scrollbar appears.
- When 17-24 emojis are configured, the menu has three rows, and no scrollbar appears.
- When 25 or more emojis are configured, the menu has three rows, and a scrollbar appears.



Configure the v2 emoji menu by passing a string containing emoji into the WebChat configuration or through localization.

Important

If you define an emoji list in the WebChat configuration, it will override any emoji lists defined in localization files.

You configure the emoji list by specifying a string of emoji characters, like "👉👉👉". WebChat will parse this string and arrange them in the emoji menu.

[illegible]

Add emoji display names

You can also add names to emojis so that their names will appear when you hover over them. To add a name to an emoji, simply add a colon after the question mark symbol, and then type the name. Separate each name with a semicolon.

The format is ;:name;

You can only add one name to an emoji. The following sample shows the format for configuring several emojis.

```
// Configure an emoji list with emoji names
_genesys.widgets.webchat.emojiList = "🌟:Star-Struck;🤪:Zany Face;👋:Face With Hand Over Mouth;😱:Shushing Face;👁️:Face With Raised Eyebrow;👁️:Bitcoin;🤮:Face Vomiting;💣:Exploding Head;👁️:Face With Monocle;👁️:Face With Symbols on Mouth;👁️:Orange Heart;💕:Love-You Gesture;👐:Palms Up Together;🧠:Brain;👦:Child;👤:Person;👤:Man;👤:Beard;👤:Older Person;👤:Woman With Headscarf;👤:Breast-Feeding;👤:Mage;👤:Fairy;👤:Vampire;👤:Merperson;👤:Elf;👤:Genie;👤:Zombie;👤:Person in Steamy Room;👤:Person Climbing;👤:Person in Lotus Position;👤:Zebra;👤:Giraffe;👤:Hedgehog;👤:Sauropod;👤:T-Rex;👤:Cricket;🥥:Coconut;🥦:Broccoli;🥨:Pretzel;🥩:Cut of Meat;🇦🇺:Australia Day;🏰:Bastille
```

```
Day;🎂:Birthday;🛍️:Black Friday;🇨🇦:Canada Day;🎪:Carnival;🧧:Chinese New Year;🎄:Christmas;
🇲🇽:Cinco de Mayo;🏮:Diwali;🛶:Dragon Boat Festival;🐣:Easter;🎬:Emoji Movie;🍂:Fall/Autumn;
👨👦:Father's Day;🎊:Festivus;🎓:Graduation;👤:Guy Fawkes;🎃:Halloween;🕎:Hanukkah;
💖:Hearts;🌞:Holi;🇵🇰:Independence Day;👩👦:Mother's Day;🎆:New Year's Eve;🏅:Olympics;
👑:Pride;👑:Queen's Birthday;🌙:Ramadan;🌸:Spring;🇮🇪:St Patrick's Day;☀️:Summer;
🏏:SuperBowl;🦃:Thanksgiving;💕:Valentine's Day;💍:Wedding / Marriage;❄️:Winter
Olympics;🏆:World Cup;🌐:World Emoji Day;";
```

Partially named lists

You don't have to add names for every emoji. You can add titles to only a select few.

[illegible]

Localization

Emojis can be localized so that each language has a preferred set of emojis and emoji titles.

Important

If you define an emoji list in the WebChat configuration, it will override any emoji lists defined in localization files.

The key name for defining an emoji list is "EmojiList". Emoji lists are defined in a localization file using the same syntax as the WebChat configuration.

```
{
  "en": {
    "webchat": {
      "EmojiList": "🌟:Star-Struck;🤪:Zany Face;🤫:Face With Hand Over Mouth;🤦:Shushing
Face;"
    }
  }
}
```

Terminate Chat session on contact side

To prevent a contact from sending another chat message using the Widget after the chat session is terminated in Designer, you must add a customization to the widget to notify it to close.

First, set up a text message informing the contact that the chat is terminated by using a Play Message Block.

Properties - Play Message



This block is used to play audio messages. These messages can be TTS (Text to Speech), Audio Files (previously uploaded in Audio Resources page, or variables played as TTS.



Prompts




Message Settings

Prompts

☒ Disable barge-in ?

☒ Always play prompt and disable buffering ?

+ Add Prompt

Type	Var?	Value	Play as	Actions
TTS	<input type="checkbox"/>	Thanks for contacting us. Goodbye!	text	  

Next, set up the Widget Register Handler for **WebChatService.messageReceived** (or look for the **messageAdded** event) to get notifications about messages received, then send the **endChat** command when the text message is received. For information about Genesys Widgets events and commands, refer to Genesys Widget API Events and Genesys Widget API Commands.

Finally, add the following customized script:

```

window._genesys.widgets.onReady = function(CXBus){
    var oWH = CXBus.registerPlugin("WebChatHandler");
    oWH.subscribe("WebChatService.messageReceived", function (e) {
        if(e.data) {
            /**
             * Extract the sample data (can be the Playback message configured in Designer)
             * and look for a specific condition to end the chat
             */
            const {messages} = e.data || {};
            let sPlayMessage = (messages) ? messages.find(message => message.type ==
'Message' && message.text == 'play message') : "";
            if(sPlayMessage) {
                oWH.command("WebChatService.endChat");
                /**
                 * Check for the chat session data stored in localStorage and clear it
                 */
                (window.localStorage.getItem("WebChatSessionData")) ?
window.localStorage.removeItem("WebChatSessionData") : "";
            }
        }
    });
};

```

Engage

Contents

- [1 Overview](#)
 - [1.1 Usage](#)
 - [1.2 Namespaces](#)
 - [1.3 Screenshots](#)
- [2 Configuration](#)
- [3 Localization](#)
- [4 API commands](#)
 - [4.1 invite](#)
 - [4.2 Example](#)
 - [4.3 Options](#)
 - [4.4 Resolutions](#)
 - [4.5 offer](#)
 - [4.6 Example](#)
 - [4.7 Options](#)
- [5 API events](#)
 - [5.1 Interaction Lifecycle](#)
 - [5.2 Lifecycle scenarios](#)
- [6 Metadata](#)
 - [6.1 Reference](#)

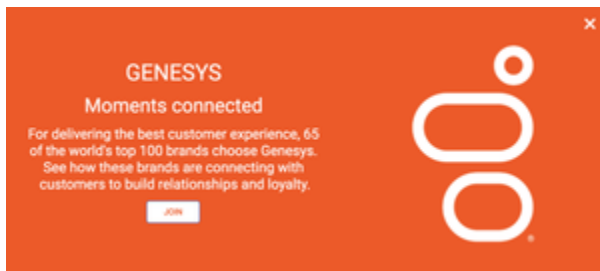
- Developer

Learn how to use the Genesys Multicloud CX plugin to integrate any Engage solution with Genesys Widgets.

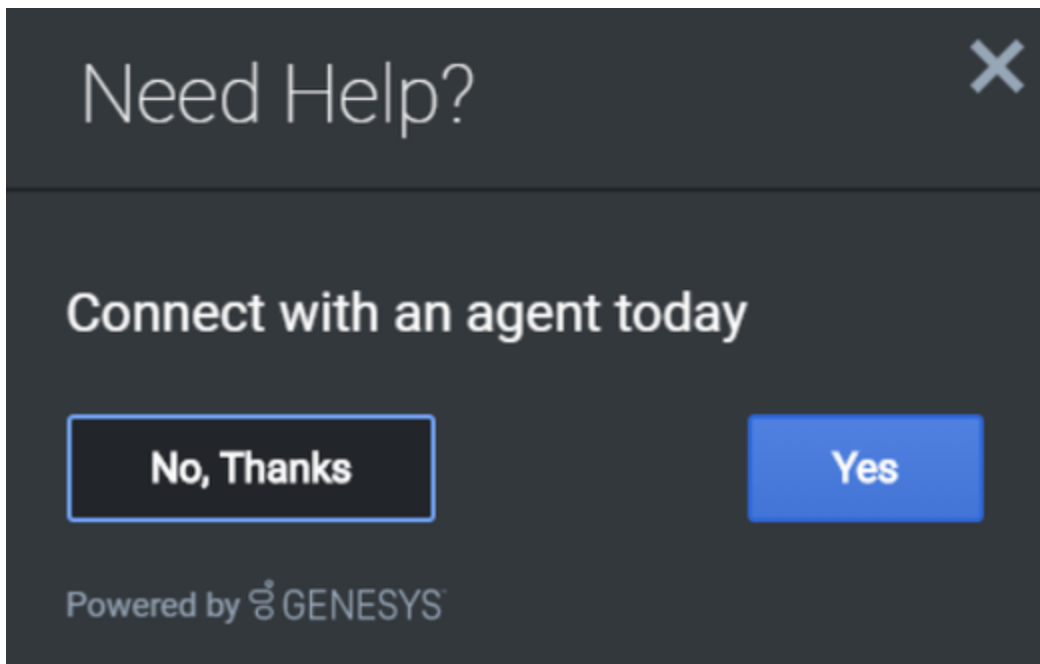
Related documentation:

-

Overview



The Genesys Multicloud CX plugin is generic and contains commands that automate customer engagement within Genesys Widgets. Starting with version 9.0.015.11, the Engage plugin includes Offers, which allows a customer to view a product or promotion on a page. It comes with many display modes and rendering options, such as overlay/toaster mode with text or image-only layouts, or both.



Usage

Use the Engage plugin to show either an invite or an offer via the following methods:

- Calling the Engage.invite command
- Calling the Engage.offer command

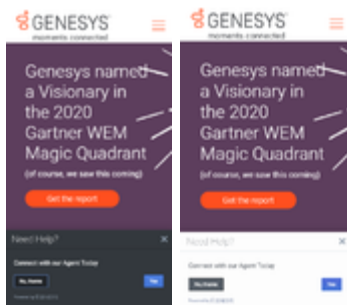
Namespaces

The Engage plugin uses the following namespaces.

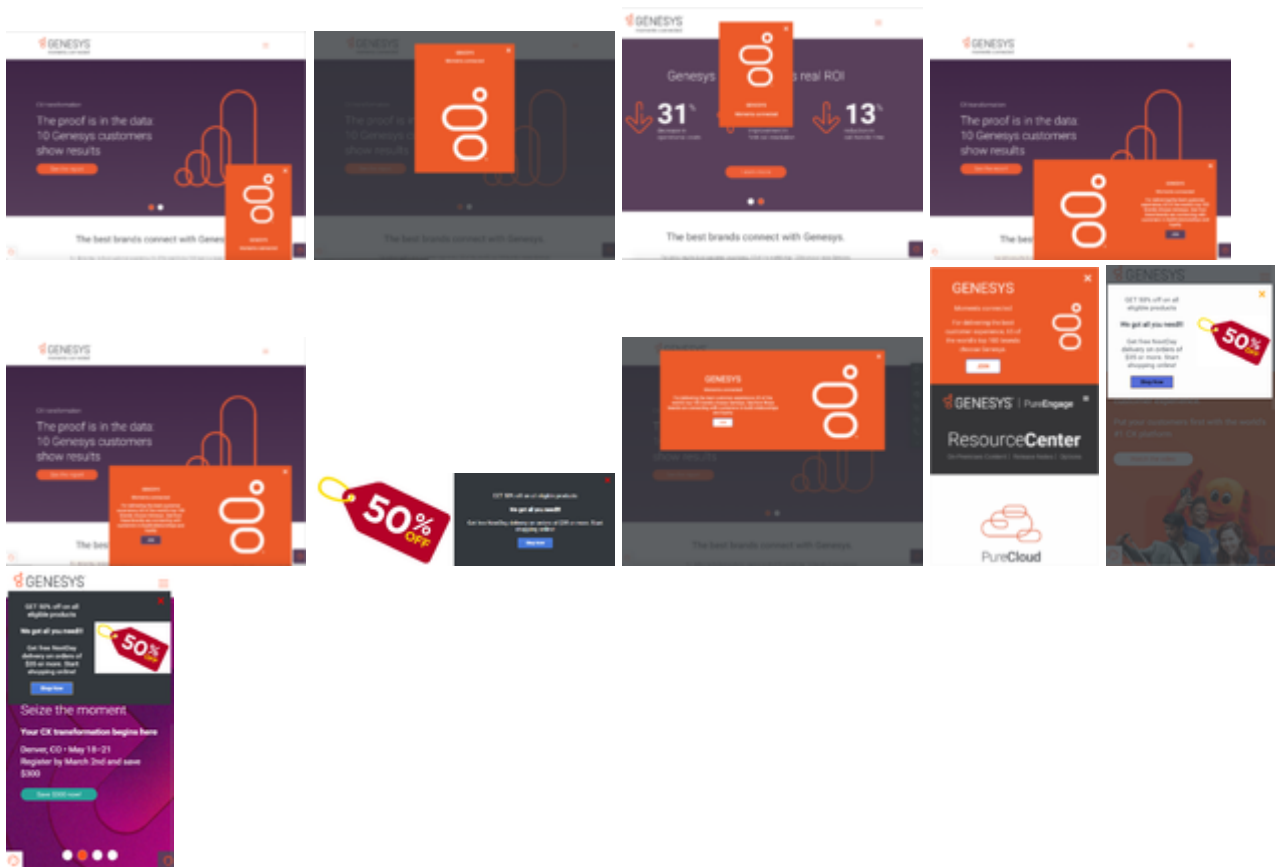
Type	Namespace
i18n - Localization	Engage
CXBus - API commands & API events	Engage
CSS	.cx-engage

Screenshots

Engage Invite



Engage Offer



Configuration

The Genesys Multicloud CX plugin doesn't have any configuration options.

Localization

The Genesys Multicloud CX plugin doesn't have any localization options.

API commands

Once you've registered your plugin on the bus, you can call commands on other registered plugins. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.command('Engage.invite');
```

invite

Opens the Engage Widget and renders the text based on the options provided. If no options are provided, the widget doesn't open.

Example

```
oMyPlugin.command('Engage.invite', {  
  'type': 'toast',  
  'timeout': 3000,  
  'title': 'Engage Title',  
  'ariaTitle': 'Engage Invite',  
  'body': 'Engage invite body content',  
  'accept': 'Yes',  
  'decline': 'No, thanks',  
  'ariaAccept': 'Yes',  
  'ariaDecline': 'No, thanks',  
  'ariaClose': 'Close',  
  'command': 'WebChat.open',  
  'options': {'proactive': true, 'userData': {'category': 'shoes'}}  
});  
  
oMyPlugin.command('Engage.invite', {  
  'type': 'toast',  
  'timeout': 3000,  
  'force': true,  
  'title': 'Engage Title',  
  'ariaTitle': 'Engage Invite',  
  'body': 'Engage invite body content',  
  'accept': 'Yes',  
  'decline': 'No, thanks',  
  'ariaAccept': 'Yes',  
  'ariaDecline': 'No, thanks',  
  'ariaClose': 'Close'  
});
```

```
}).done(function(response){  
    // Act upon the received response code  
    switch(response){  
    case 'accepted':oMyPlugin.command('WebChat.open');  
        break;  
    case 'declined': break;  
    case 'closed': break;  
    case 'timeout': break;  
    }  
});
```

Options

Option	Type	Description	Accepted values	Default	Introduced/ updated
type	string	Widget display type.	toast		
timeout	number	Timeout integer in milliseconds.	n/a		
title	string	String for widget title.	n/a		
ariaTitle	string	Aria label text for the Engage invite window.	n/a		9.0.015.04
body	string	String for offer body text.	n/a		
accept	string	String for Accept button text.	n/a		
ariaAccept	string	Aria label text for the Accept button.	n/a		9.0.016.10
decline	string	String for Decline button text.	n/a		
ariaDecline	string	Aria label text for the Decline button.	n/a		9.0.016.10
ariaClose	string	Aria label text for the Engage Close button.	n/a		9.0.016.10
command	string	Command to execute.	n/a		
options	object	Options related to the command provided.	n/a		

Option	Type	Description	Accepted values	Default	Introduced/updated
priority	number	Replace the active lower priority Engage invite with the higher priority Engage invite.	n/a	0	9.0.015.11
force	boolean	Replace the active Engage invite with the new Engage invite irrespective of priorities.	n/a	false	9.0.015.11

Resolutions

Status	When	Returns
resolved	Engage invite is accepted by user.	accepted
resolved	Engage invite is declined by user.	declined
resolved	Engage invite widget is closed by user.	closed
resolved	Engage invite widget closes due to timeout.	timeout

offer

Opens a widget for a product offer using the data sent through the command options provided below. The widget can include both rendering options and the actual data that needs to be displayed in the Offer Widget. If no options are provided, the widget will not open.

Example

```
oMyPlugin.command('Engage.offer', {  
  mode: 'overlay',  
  modal: true,  
  layout: 'leftText',  
  title: 'GRAB WHAT YOU NEED!!',  
  ariaTitle: 'Offers',  
  headline: 'We Got All!',  
  description: 'Get free NextDay delivery on orders of $35 or more. Start shopping  
now!',  
  cta: {  
    text: 'Join',  
    url: 'https://www.genesys.com',  
    target: '_blank'  
  },  
})
```

```
image:{
  src:'https://picsum.photos/id/237/300/300',
  alt:'Alternate Text for Image'
},
styles:{
  closeButton:{
    'color':'red'
  }
},
ariaCTA:'Join',
ariaClose:'Close Offer'
});
```

Options

Option	Type	Description	Accepted values	Default	Introduced/updated
mode	string	The display type of the Offer widget.	overlay, toaster	toaster	9.0.015.04
modal	boolean	Applicable only when mode is 'overlay'. A smokescreen will be shown in the background of overlay modal window. This window can be dismissed by clicking anywhere in the smokescreen area.	n/a	false	9.0.015.04
layout	string	Additional layout options are supported for all modes.	minimal, leftText, rightText, topText, bottomText	leftText	9.0.015.04
headline	string	The Offer title header text.	n/a	n/a	9.0.015.04
ariaTitle	string	Aria label text for the Offer window.	n/a	n/a	9.0.015.04
description	string	The Offer body description text.	n/a	n/a	9.0.015.04
cta	object	An object containing HTML attributes and/	n/a	n/a	9.0.015.04

Option	Type	Description	Accepted values	Default	Introduced/updated
		or CXBus commands for the CTA (call to action) button.			
cta.text	string	The CTA button text.	n/a	n/a	9.0.015.04
cta.url	string	The URL string for the CTA button. Note: The URL must be properly defined with the complete Protocol URL Address. For example, https://www.genesys.com .	_blank, _parent, _self, _top, framename	n/a	9.0.015.04
cta.target	string	Specifies where the URL is opened.	n/a	n/a	9.0.015.04
cta.command	string	A CXBus command to execute.	n/a	n/a	9.0.015.04
cta.commandOptions	string	Options related to CXBUS command.	n/a	n/a	9.0.015.04
image	object	An object containing image tag attributes.	n/a	n/a	9.0.015.04
image.src	string	The URL of the image.	n/a	n/a	9.0.015.04
image.alt	string	Alternate text for the image.	n/a	n/a	9.0.015.04
image.title	string	To indicate the screen reader user whether the image opens the URL in a new window.	n/a	n/a	9.0.016.10
insertAfter	string	Applicable only in mobile mode. An ID or class name of an HTML selector from the host page. The offer will be inserted after this	n/a	n/a	9.0.015.04

Option	Type	Description	Accepted values	Default	Introduced/updated
		element. Precede the value mentioned here with the standard Class ('.') and ID selector ('#') character.			
insertBefore	string	Applicable only in mobile mode. An ID or class name of an HTML selector from the host page. The offer will be inserted before this element. Precede the value mentioned here with the standard Class ('.') and ID selector ('#') character.	n/a	n/a	9.0.015.04
insertInto	string	Applicable only in mobile mode. An ID or class name of an HTML selector from the host page. The offer will be appended inside this element. Precede the value mentioned here with the standard Class ('.') and ID selector ('#') character.	n/a	n/a	9.0.015.04
styles	object	An object containing styles for the offer content.	n/a	n/a	9.0.015.04
styles.closeButtonobject		An object containing	n/a	n/a	9.0.015.04

Option	Type	Description	Accepted values	Default	Introduced/updated
		styles for the close button.			
styles.closeButton.color	string	The color of the close button.	n/a	n/a	9.0.015.04
styles.closeButton.opacity	number	The CSS 'opacity' property for the close button.	n/a	n/a	9.0.015.04
styles.overlay	object	An object containing styles for the overlay container.	n/a	n/a	9.0.015.04
styles.overlay.top	string	The CSS 'top' property for the overlay container.	n/a	n/a	9.0.015.04
styles.overlay.right	string	The CSS 'right' property for the overlay container.	n/a	n/a	9.0.015.04
styles.overlay.bottom	string	The CSS 'bottom' property for the overlay container.	n/a	n/a	9.0.015.04
styles.overlay.left	string	The CSS 'left' property for the overlay container. Note: When all the position values are provided, the order of precedence will be top, right, bottom, and left.	n/a	n/a	9.0.015.04
styles.overlay.center	boolean	Aligns the overlay container to the center of the screen.	n/a	true	9.0.015.04
styles.offer	object	An object containing styles for the Offer window.	n/a	n/a	9.0.015.04
styles.offer.backgroundColor	string	The background	n/a	n/a	9.0.015.04

Option	Type	Description	Accepted values	Default	Introduced/updated
		color of the offer.			
styles.offer.color	string	The text color of the offer.	n/a	n/a	9.0.015.04
styles.offer.padding	string	The padding for the offer container.	n/a	0	9.0.015.04
styles.title	object	An object containing styles for the title.	n/a	n/a	9.0.015.04
styles.title.font	string	The CSS 'font' property for the title.	n/a	n/a	9.0.015.04
styles.title.textAlign	string	The CSS 'text-align' property for the title.	n/a	n/a	9.0.015.04
styles.headline	object	An object containing styles for the header text.	n/a	n/a	9.0.015.04
styles.headline.font	string	The CSS 'font' property for the header text.	n/a	n/a	9.0.015.04
styles.headline.textAlign	string	The CSS 'text-align' property for the header text.	n/a	n/a	9.0.015.04
styles.description	object	An object containing styles for the offer description text.	n/a	n/a	9.0.015.04
styles.description.font	string	The CSS 'font' property for the description text.	n/a	n/a	9.0.015.04
styles.description.textAlign	string	The CSS 'text-align' property for the description text.	n/a	n/a	9.0.015.04
styles.ctaButton	object	An object containing styles for call to action button in the	n/a	n/a	9.0.015.04

Option	Type	Description	Accepted values	Default	Introduced/updated
		offer window.			
styles.ctaButton.font	string	The CSS 'font' property for the text in CTA button.	n/a	n/a	9.0.015.04
styles.ctaButton.textAlign	string	The CSS 'text-align' property for the text in CTA button.	n/a	n/a	9.0.015.04
styles.ctaButton.background	string	The CSS 'background' property for the CTA button.	n/a	n/a	9.0.015.04
styles.ctaButton.color	string	The CSS 'color' property for the text in CTA button.	n/a	n/a	9.0.015.04
styles.ctaButton.fontSize	string	The CSS 'font-size' property for the text in CTA button.	n/a	n/a	9.0.015.04
ariaCTA	string	Aria label text for the Offer CTA button.	n/a	n/a	9.0.016.10
ariaClose	string	Aria label text for the Offer Close button.	n/a	n/a	9.0.016.10
priority	number	Replace the active lower priority Engage Offer with the higher priority Engage Offer.	n/a	0	9.0.015.11
force	boolean	Replace the active Engage Offer with the new Engage Offer irrespective of priorities.	n/a	false	9.0.015.11

API events

Once you've registered your plugin on the bus, you can subscribe to and listen for published events. Here's how to use the global bus object to register a new plugin on the bus.

Important

The global bus object is a debugging tool. When implementing widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Genesys Widgets extensions for more information about extending Genesys Widgets.

```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');  
oMyPlugin.subscribe('Engage.ready', function(e){});
```

Name	Description	Data	Introduced/updated
ready	The Engage widget is initialized and ready to accept commands on the bus.	n/a	
opened	The Engage widget opens. Note: Applicable only to Engage.offer command	Metadata	9.0.015.04
CTA	When the user clicks the CTA button in the Engage widget. Note: Applicable only to Engage.offer command	Metadata	9.0.015.04
hover	When the user first hovers over the Engage widget. Note: Applicable only to Engage.offer command	Metadata	9.0.015.04
dismissed	When the user closes the Engage widget by clicking the Close button. Note: Applicable only to Engage.offer command	Metadata	9.0.015.04
closed	The Engage widget closes. Note: Applicable only to Engage.offer command	Metadata	9.0.015.04

Important

Applicable only for Engage.offer command.

Interaction Lifecycle

Every offer interaction has a sequence of events we describe as the *Interaction Lifecycle*. These events track progress and user choices from the beginning of an interaction (opening Offers), to the end (closing Offers), and every step in between.

The following events comprise the Interaction Lifecycle:

```
ready  
opened  
CTA  
hover  
dismissed  
closed
```

Lifecycle scenarios

An Interaction Lifecycle can vary based on each user's intent and experience with the Offer widget. Here are several sequences of events in the lifecycle that correspond to different scenarios.

The user opened the Offer widget but changed their mind and closed it without seeing the offer details:

```
ready -> opened -> dismissed -> closed
```

The user opened the Offer widget, hovered over the offer details, and then closed it:

```
ready -> opened -> hover -> dismissed -> closed
```

The user opened the Offer widget and clicked on the button, which triggers CTA:

```
ready -> opened -> CTA -> closed
```

Tip

For a list of all Offer events, see API events.

Metadata

Each event in the Interaction Lifecycle includes the following block of metadata. By default, all values are set to false. As the user progresses through the lifecycle of an Offer Engage interaction, these values are updated.

The metadata block contains Boolean state flags, timestamps, and elapsed times. These values can be used to track and identify trends or issues with interactions. During runtime, the metadata can help you offer a smart and dynamic experience to your users.

Reference

Name	Type	Description	Introduced/updated
opened	integer (timestamp)	Timestamp indicating when the offer was opened.	9.0.015.04
closed	integer (timestamp)	Timestamp indicating when the offer was closed.	9.0.015.04
dismissed	integer (timestamp)	Timestamp indicating when the user dismissed the offer by clicking the close button.	9.0.015.04
triggeredCTA	integer (timestamp)	Timestamp indicating when the CTA was triggered.	9.0.015.04
timeBeforeCTA	integer (milliseconds)	Total time in milliseconds from when the user opened the offer to when the CTA is triggered.	9.0.015.04
timeFirstHover	integer (timestamp)	Timestamp indicating when the user first hovered over the offer.	9.0.015.04
timeBeforeHover	integer (milliseconds)	Total time in milliseconds from when the user opened the offer to when the user first hovered over the offer.	9.0.015.04
timeElapsedHover	integer (milliseconds)	Total time in milliseconds when the user hovered over the offer.	9.0.015.04
elementClicked	string	Name of CTA element that was clicked.	9.0.015.04

Genesys Widgets extensions

Contents

- [1 Overview](#)
- [2 Defining extensions](#)
- [3 Creating a new CXBus plugin](#)
- [4 Use cases](#)
 - [4.1 Example: subscribing to an event](#)
 - [4.2 Example: publishing an event](#)
 - [4.3 Example: calling a command](#)
 - [4.4 Example: registering a command](#)
 - [4.5 Example: using the 'before\(\)' method](#)

- Developer

Learn how to create your own plugins and widgets.

Related documentation:

-

Overview

Genesys Widgets allows you to create your own plugins and widgets. These extensions are an easy way to define your own functionality, while using the same resources as the core Genesys Widgets.

Defining extensions

Extensions are defined at runtime before Genesys Widgets loads. 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.

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 the 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 namespace.

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
});
```

Example: using the 'before()' method

Allows you to set up an interrupt that is executed before a command every time that command is called. With this feature, you can link execution of a command with other logic, modify command options before they're used, or cancel execution of a command.

You can specify multiple "before" functions for a single command. They will be executed in order with the output of one providing the input to the next. If one of the functions does not return an object, execution will stop and the command will be cancelled.

```
oTestExtension.before("WebChat.open", function(oData){
    // oData == the options passed into the command call
    // e.g. if this command is called: oMyPlugin.command("WebChat.open", {form: {firstname:
    "Mike"}}});
    // then oData will == {form: {firstname: "Mike"}}

    // You must return oData back, or an empty object {} for execution to continue.
    // If you return false|undefined|null, execution of the command will be stopped
    return oData;
});
```

Genesys Widgets videos

Contents

- [1 Introduction to Widgets](#)
- [2 Getting started with the Genesys WebChat Widget](#)
- [3 WebChat features](#)
- [4 The Callback Widget](#)
- [5 The CallUS Widget](#)

This collection of videos from the Genesys Vimeo channel demonstrates some of the most commonly used features of Genesys Widgets.

Related documentation:

-

Introduction to Widgets

Available Widgets on Genesys Multicloud CX

[Link to video](#)

Getting started with the Genesys WebChat Widget

How to configure Genesys Widgets and start using WebChat

[Link to video](#)

WebChat features

Features of the WebChat Widget

[Link to video](#)

The Callback Widget

Features of the Callback Widget

[Link to video](#)

The CallUS Widget

Features of the CallUs Widget

[Link to video](#)