



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 Reference

Genesys Widgets 8.5

Table of Contents

Genesys Widgets Reference	6
WebChatService	7
Configuration	8
Localization	11
API Commands	12
API Events	28
WebChat	31
Configuration	38
Localization	41
API Commands	45
API Events	57
Metadata	58
Customizable Chat Registration Form	61
SendMessageService	68
Configuration	69
Localization	71
API Commands	72
API Events	76
SendMessage	77
Configuration	82
Localization	84
API Commands	86
API Events	93
Metadata	94
GWE	96
Configuration	97
Localization	99
API Commands	100
API Events	103
CoBrowse	104
Configuration	105
Localization	106
API Commands	107
API Events	111
App	112

Configuration	113
Localization	117
API Commands	118
API Events	123
Calendar	124
Configuration	127
Localization	130
API Commands	132
API Events	137
CallbackService	138
Configuration	139
Localization	140
API Commands	141
API Events	146
Callback	147
Configuration	151
Localization	153
API Commands	155
API Events	159
Metadata	160
CallUs	162
Configuration	168
Localization	170
API Commands	171
API Events	174
ChannelSelector	175
Configuration	183
Localization	187
API Commands	188
API Events	194
ChatDeflection	195
Configuration	202
Localization	204
API Commands	206
API Events	209
Common	210
Console	230

Configuration	232
Localization	233
API Commands	234
API Events	237
KnowledgeCenterService	238
Configuration	240
Localization	242
API Commands	243
API Events	254
Overlay	255
Configuration	257
Localization	258
API Commands	259
API Events	262
Search	263
Configuration	269
Localization	271
API Commands	273
API Events	279
SideBar	280
Configuration	285
Localization	288
API Commands	289
API Events	294
StatsService	295
Configuration	296
Localization	297
API Commands	298
API Events	301
Toaster	302
Configuration	304
Localization	305
API Commands	306
API Events	309
WindowManager	310
Configuration	312
Localization	313

API Commands	314
API Events	316

Genesys Widgets Reference

The Widgets Reference covers all commands, events, configuration, and localization details for each widget.

- [WebChatService](#)
- [WebChat](#)
- [SendMessageService](#)
- [SendMessage](#)
- [GWE](#)
- [CoBrowse](#)
- [App](#)
- [Calendar](#)
- [CallbackService](#)
- [Callback](#)
- [CallUs](#)
- [ChannelSelector](#)
- [ChatDeflection](#)
- [Common](#)
- [Console](#)
- [KnowledgeCenterService](#)
- [Overlay](#)
- [Search](#)
- [SideBar](#)
- [StatsService](#)
- [Toaster](#)
- [WindowManager](#)

WebChatService

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

WebChatService exposes a high-level API for utilizing Genesys chat services. You can use these services for monitoring and modifying a chat session on the front-end or for developing your own custom WebChat widgets. Rather than developing a custom chat UI and using the chat REST API, using WebChatService drastically simplifies integration and greatly improves reliability, features, and compatibility on the bus for all widgets.

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

WebChat Service plugin has the following namespaces tied-up with 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 meant as a plug-n-play type of plugin and works as-is.

Configuration

Description

WebChat and WebChatService share the configuration namespace '_genesys.widgets.webchat'. WebChat has UI options while WebChatService has connection options.

Example

```
window._genesys.widgets.webchat = {  
  apikey: 'n3eNkgLLgLKXREBMjGm6lygOHHOK8VA',  
  dataURL: 'https://api.genesyscloud.com/gms-chat/2/chat',  
  
  cometD: {  
    enabled: false,  
    cometURL: 'http://host:port/genesys/cometd',  
    channel: '/service/chatV2/customer-support',  
    apiURL: 'http://host:port/genesys/2/chat-ntf',  
    websocketEnabled: true,  
    logLevel: 'info'  
  },  
  
  userData: {},  
  emojis: true,  
  actionsMenu: true,  
  
  autoInvite: {  
    enabled: false,  
    timeToInviteSeconds: 10,  
    inviteTimeoutSeconds: 30  
  },  
  
  chatButton: {  
    enabled: true,  
    template: '<div>CHAT NOW</div>',  
    effect: 'fade',  
    openDelay: 1000,  
    effectDuration: 300,  
    hideDuringInvite: true  
  }  
};
```


Options

Name	Type	Description	Default	Required
apikey	string	Apigee Proxy secure token.	n/a	Yes, if using Apigee Proxy
endpoint	string	Manually select the endpoint to initiate chat on.	n/a	
dataURL	string (URL)	URL for GMS REST chat service. If cometD.enabled is set to true, this property will be ignored.	n/a	Always
cometD	object	Object container for CometD configuration options.	{enabled: false, cometURL: , channel: '/service/chatV2/customer-support', websocketEnabled: true, logLevel: 'info'}	Yes, if using CometD
cometD.enabled	boolean	Enables or disables CometD connection method. If set to false or left undefined, WebChatService will connect to REST services through the dataURL specified.	false	Yes, if using CometD
cometD.cometURL	string (URL)	URL for GMS CometD connection. cometD.enabled must be set to true for WebChatService to connect to this service.	n/a	Yes, if using CometD
cometD.channel	string (path)	CometD channel for receiving chat messages.	'/service/chatV2/customer-support'	Yes, if using CometD
cometD.apiURL	string (URL)	URL for additional CometD services such as file upload and download.	n/a	Yes, if using CometD with file uploads
cometD.websocketEnabled	boolean	If set to true, CometD will attempt to connect	true	

Name	Type	Description	Default	Required
		through websockets. If set to false, CometD will only use long-polling. CometD will fallback to long-polling if it can't connect via websockets.		
cometD.logLevel	string	Sets the log level for the CometD library. Values are 'warn', 'info', or 'debug'.	'info'	
userData	object	Arbitrary JSON attached data to include when initiating a chat.	{}	
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout.	3000	
xhrFields	object	Allows you to set the properties for the AJAX xhrFields object (e.g. {withCredentials: false}).	{withCredentials: false}	
pollExceptionLimit	number	Number of successive poll exceptions (chat server offline) before WebChatService publishes 'chatServerWentOffline'.	5	
restoreTimeout	number	Number of milliseconds before restore timeout. Prevents the chat session from restoring after a certain time away from the session (e.g. user navigated to a different site during chat and never ended the session).	60000	

Localization

No Localization options

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('WebChatService.getAgents');
```

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 GMS. Intended to be used by WebChat widgets only. Should not be invoked manually.

Example

```
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  
});
```

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	When server confirms session started	(AJAX Response Object)
rejected	When a chat session is already active	'There is already an active chat session'
rejected	When AJAX exception occurs	(AJAX Response Object)
rejected	When server exception occurs	(AJAX Response Object)
rejected	When userData is invalid	'malformed JSON provided in userData property'

endChat

Ends the chat session with the chat server via GMS. Intended to be used by WebChat widgets only. Should not be invoked manually.

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	When active session is ended successfully	(AJAX Response Object)
rejected	If no chat session is currently active	'There is no active chat session'

sendMessage

Send a message from the client to the chat session. Intended to be used by WebChat widgets only. Should not be invoked manually.

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

sendCustomNotice

Send a custom notice from the client to the chat server.

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
resolved	When message is successfully sent	(AJAX Response Object)
rejected	When AJAX exception occurs	(AJAX Response Object)

sendTyping

Send 'customer typing' notification to chat session. A visual indication will be shown to agent. Intended to be used by WebChat widgets only. Should not be invoked manually.

Example

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

Resolutions

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

sendFilteredMessage

Send 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	When there is an active session	n/a
rejected	If no chat session is currently active	'No active chat session.'

addPrefilter

Add a new regular expression prefilter to the prefilter list. Any messages matched using the prefilters 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	When valid filters are provided	Array of all registered prefilters
rejected	When 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 JSON you want to send to the server for this active session.

Resolutions

Status	When	Returns
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

poll

Internal use only. Start polling for new messages. Intended to be used by WebChat widgets only. Should not be invoked manually.

Example

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

Resolutions

Status	When	Returns
resolved	When there is an active session	n/a

Status	When	Returns
rejected	WebChatService isn't calling this command	'Access Denied to private command. Only WebChatService is allowed to invoke this command'
rejected	If no chat session is currently active	'previous poll has not finished'

startPoll

Start automatic polling for new messages. Intended to be used by WebChat widgets only. Should not be invoked manually.

Example

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

Resolutions

Status	When	Returns
resolved	When there is an active session	n/a
rejected	When no chat session is currently active	No active chat session.
rejected	When CometD is enabled	Polling is not supported when using CometD

stopPoll

Stop automatic polling for new messages. Intended to be used by WebChat widgets only. Should not be invoked manually.

Example

```
oMyPlugin.command('WebChatService.stopPoll').done(function(e){
```

```
        // WebChatService stopped polling successfully
    }).fail(function(e){
        // WebChatService failed to stop polling
    });
```

Resolutions

Status	When	Returns
resolved	When there is an active session	n/a
rejected	If no chat session is currently active	No active chat session.

resetPollExceptions

Reset 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
resolved	Always	n/a
rejected	Never	undefined

restore

Internal use only. Intended to be used by WebChatService only. Should not be invoked manually.

Example

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

Resolutions

Status	When	Returns
resolved	Session has been found.	n/a
rejected	Session can not be found	n/a

getTranscript

Fetch an array of all messages in the chat session

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 that have participated 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

Return 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),

Status	When	Returns
		duration: (int time)}

sendFile

Sends the file from the client machine to the agent.

Example

```
oMyPlugin.command('WebChatService.sendFile', {files: $('<input/>').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 <input type="file"/>)

Resolutions

Status	When	Returns
resolved	When the file sent is a valid type and size	(AJAX Response Object)
rejected	When the file sent is an invalid type	(AJAX Response Object)
rejected	When the number of uploads is exceeded	(AJAX Response Object)
rejected	When the file size exceeds the limit	(AJAX Response Object)
rejected	When the file size is too large or an unknown error occurs	(AJAX Response Object)
rejected	When CometD is enabled	File Uploads are not currently supported when using CometD

downloadFile

Downloads the file to the client machine.

Example

```
oMyPlugin.command('WebChatService.downloadFile', {fileId: '1'}).done(function(e){  
    // WebChatService sent file successfully  
}).fail(function(e){  
    // WebChatService failed to send file  
});
```

Options

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

getFileLimits

This optional request can be used before uploading a large file. If size, type, or other constraints are not met, then uploading the file will fail, avoiding network and CPU overhead.

Example

```
oMyPlugin.command('WebChatService.getFileLimits').done(function(e){  
    // WebChatService got file limits successfully  
}).fail(function(e){  
    // WebChatService failed to get file limits  
});
```

Resolutions

Status	When	Returns
resolved	When the file limits request succeeds	(AJAX Response Object)
rejected	When the file limits request fails	(AJAX Response Object)
rejected	When CometD is enabled	File Uploads are not currently supported when using CometD

registerTypingPreviewInput

Select an HTML input to watch for key events. Used to trigger startTyping and stopTyping automatically. Intended to be used by WebChat widgets only. Should not be invoked manually.

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	When valid HTML input reference is provided	n/a
rejected	When 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

Allows you to register a function that receives the message JSON, allowing you to manipulate the values before it is rendered in the transcript.

Example

```
oMyPlugin.command('WebChatService.registerPreProcessor', {preprocessor:
function({})}).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	When a valid preprocessor function is provided and is registered.	The registered preprocessor function.
rejected	When an invalid preprocessor function is provided.	No preprocessor function provided. Type provided was '<DATATYPE>'.

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('WebChatService.ready', function(e){});
```

Name	Description	Data
ready	WebChatService is initialized and ready to accept commands.	n/a
restored	Chat session has been restored after page navigation or refresh.	n/a
restoreTimeout	Chat session restoration attempted was denied after user navigated away from originating website for longer than the time limit: default 60 seconds.	n/a
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.	n/a
error	An error occurred between the client and the server.	(AJAX Response)
started	Chat session has successfully	n/a

Name	Description	Data
	started.	
ended	Chat session has successfully ended.	n/a
agentTypingStarted	Agents has started typing a new message.	n/a
agentTypingStopped	Agent has stopped typing.	n/a
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: (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)}
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 configuration option 'pollExceptionLimit' has been	n/a

Name	Description	Data
	exceeded. Default limit is 5 poll exceptions. '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.	
chatServerBackOnline	Chat server had come back online after going offline. This will only be published after 'chatServerWentOffline'.	n/a
connectionPending	If there is a connection problem and WebChatService is trying to reconnect, this event will be published. Published before 'chatServerWentOffline'.	n/a
connectionRestored	Is published when the connection has been reestablished. Publishes at the same time as 'chatServerBackOnline'.	n/a

WebChat

 Live Chat 


**will be with you shortly. In
meantime, can I assist you with
any questions you may have?
Please type a question into the
input field below.**



2:37 PM


agent will be with you shortly ...

2:37 PM

John Doe
Hi
2:37 PM



Powered by  GENESYS

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

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. Customers can also initiate a Co-browse session with the agent directly from WebChat (Co-browse license and configuration required). Other features include minimize/maximize, auto-reconnect, and a built-in invite feature.

Usage

WebChat can be launched manually by the following methods:

- Calling the **command** "WebChat.open"
- Configuring **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)

Deployment Notes

WebChat Configuration

Genesys WebChat utilizes the Genesys Mobile Services (GMS) Chat API v2. For the purposes of chat, GMS can be installed in Chat-only mode (without Cassandra).

Chat Service Configuration in GMS

In order to configure your chat service in GMS, please follow these [instructions](#).

Important

The GMS configuration section referring to your chat service must follow the Chat v2 conventions. For example, if you want a chat service called "mychatservice", your configuration section must be called "chat.mychatservice" (not

"service.mychat-service", as was the case for Chat v1 services).

For more information on configuring chat support in GMS, please see the following links:

- [Chat API Version 2](#)
- [Setting Chat Dependencies](#)
- [Configuration Options Reference](#)

Can I modify the Chat Registration Form?

Yes, Chat Registration Form is customizable by defining your own form elements, thus bypassing the default registration form. For implementation, see [Customizable Chat Registration Form](#).

Customization

All static text shown in the WebChat Widget are fully customizable and **localizable** by adding entries into your **configuration** and **localization** options.

WebChat supports Themes. You may create and register your own themes for Genesys Widgets.

Namespace

WebChat plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	webchat
i18n - Localization	webchat
CXBus - API Commands & API Events	WebChat
CSS	.cx-webchat

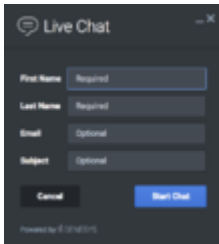
Mobile Support

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 fullscreen templates that are optimized for both portrait and landscape orientations.

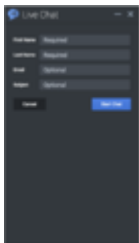
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

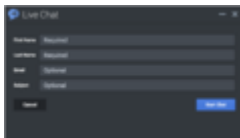
"Dark" Theme



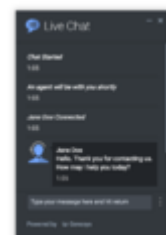
Desktop docked view showing form



Mobile fullscreen view in portrait orientation showing form



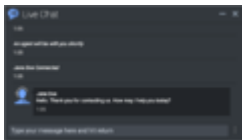
Mobile fullscreen view in Landscape orientation showing form



Desktop docked view showing transcript



Mobile fullscreen view in portrait orientation showing transcript



Mobile fullscreen view in landscape orientation showing transcript

"Light" Theme



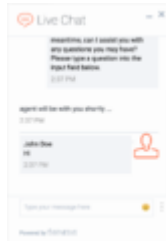
Desktop docked view showing form



Mobile fullscreen view in portrait orientation showing form



Mobile fullscreen view in Landscape orientation showing form



Desktop docked view showing transcript



Mobile fullscreen view in portrait orientation showing transcript



Mobile fullscreen view in landscape orientation showing transcript

Configuration

Description

WebChat and WebChatService share the configuration namespace '_genesys.widgets.webchat'. WebChat has UI options while WebChatService has connection options.

Example

```
window._genesys.widgets.webchat = {  
  apiKey: 'n3eNkgLLgLKXREBMjGm6lygOHHOK8VA',  
  dataURL: 'https://api.genesyscloud.com/gms-chat/2/chat',  
  userData: {},  
  emojis: false,  
  uploadsEnabled: false,  
  confirmFormCloseEnabled: true,  
  actionsMenu: true,  
  maxMessageLength: 140,  
  
  autoInvite: {  
    enabled: false,  
    timeToInviteSeconds: 10,  
    inviteTimeoutSeconds: 30  
  },  
  
  chatButton: {  
    enabled: true,  
    template: <div class='cx-icon' data-icon='chat'></div>,  
    effect: 'fade',  
    openDelay: 1000,  
    effectDuration: 300,  
    hideDuringInvite: true  
  }  
};
```

Options

Name	Type	Description	Default	Required
emojis	boolean	Enable/disable Emoji menu inside chat message input.	false	
uploadsEnabled	boolean	Show/Hide the	false	

Name	Type	Description	Default	Required
		Send File button. The button will be shown if the value is set to true.		
confirmFormCloseEnabled	boolean	Enable or disable displaying a confirmation message before closing WebChat if information has been entered into the registration form.	true	
timeFormat	number/string	This sets the time format for the timestamps in this widget. It can be 12 or 24.	12	
actionsMenu	boolean	Enable/disable actions menu next to chat message input.	true	
maxLength	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. By default, no limit is imposed by the client.	n/a	
autoInvite.enabled	boolean	Enable/disable auto-invite feature. Automatically invites user to chat after user idles on page for preset time.	false	
autoInvite.timeToInvite	seconds	Number of seconds of idle time before inviting customer to chat.	5	
autoInvite.inviteTimeout	seconds	Number of seconds to wait, after showing invite, before closing chat invite.	30	
chatButton.enabled	boolean	Enable/disable chat button on	false	

Name	Type	Description	Default	Required
		screen.		
chatButton.template	string	Custom HTML string template for chat button.	<pre><div class='cx- widget cx- webchat-chat- button' data- message='ChatButton' data-gcb- service- node='true'><div class='cx-icon' data- icon='chat'></div></div></pre>	
chatButton.effect	string	Type of animation effect when revealing chat button. 'slide' or 'fade'.	fade	
chatButton.openDelay	number	Number of milliseconds before displaying chat button on screen.	1000	
chatButton.effectDuration	number	Length of animation effect in milliseconds	300	
chatButton.hideDuringInvite	boolean	When auto-invite feature is activated, hide the chat button. When invite is dismissed, reveal the chat button again.	true	

Localization

Customer Defined Strings

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 `<%Agent%>` special value. When used, the agent's name is rendered in its place at runtime.

Error Handling

Customers can define their own error messages by defining them 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 setup localization, please refer to the [Localization Guide](#)

Usage

'webchat' namespace should be used when defining localization strings for WebChat plugin in your i18n JSON file.

In the example below, we demonstrate defining new strings for the 'en' (English) language. You may 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

Find more information on configuring Inactivity Monitoring for Chat Server [here](#)

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": "<%Agent%> Connected",
      "AgentDisconnected": "<%Agent%> Disconnected",
      "SupervisorNameDefault": "Supervisor",
      "SupervisorConnected": "<%Agent%> Connected",
      "SupervisorDisconnected": "<%Agent%> Disconnected",
      "AgentTyping": "...",
      "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",
    "ChatFormCancel": "Cancel",
    "ChatFormClose": "Close",
    "ChatInputPlaceholder": "Type your message here",
    "ChatInputSend": "SEND",
    "ChatEndQuestion": "Are you sure you want to end this chat session?",
    "ChatEndCancel": "Cancel",
    "ChatEndConfirm": "End chat",
    "ConfirmCloseWindow": "Are you sure you want to close chat?",
    "ConfirmCloseCancel": "Cancel",
    "ConfirmCloseConfirm": "Close",
    "ActionsDownload": "Download transcript",
    "ActionsEmail": "Email transcript",
    "ActionsPrint": "Print transcript",
    "ActionsCobrowseStart": "Start Co-browse",
    "ActionsSendFile": "Send File",
    "ActionsCobrowseStop": "Exit Co-browse",
    "ActionsVideo": "Invite to Video Chat",
    "ActionsTransfer": "Transfer",
    "ActionsInvite": "Invite",
    "InstructionsTransfer": "Open this link on another device to transfer
your chat session</br></br><%link%>",
    "InstructionsInvite": "ShowSurveyAcceptare this link with another
person to add them to this chat session</br></br><%link%>",
    "InviteTitle": "Need help?",
    "InviteBody": "Let us know if we can help out.",
    "InviteReject": "No thanks",
    "InviteAccept": "Start chat",
    "ChatError": "There was a problem starting the chat session. Please
Retry.",
    "ChatErrorButton": "OK",
    "DownloadButton": "Download",
    "FileSent": "has sent:",
    "FileTransferRetry": "Retry",
    "FileTransferError": "OK",
    "FileTransferCancel": "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",
    "EndConfirmBody": "Would you really like to end your chat session?",
    "EndConfirmAccept": "End chat",
    "EndConfirmReject": "Cancel",
    "SurveyOfferQuestion": "Would you like to participate in a survey?",
    "ShowSurveyAccept": "Yes",
    "ShowSurveyReject": "No",
    "UnreadMessagesTitle": "unread",
    "AriaYouSaid": "You said",
    "AriaSaid": "said",
    "AriaSystemSaid": "System said",
    "AriaMinimize": "Live Chat minimize",
    "AriaMaximize": "Live Chat Maximize",
    "AriaClose": "Live chat close",
    "Errors": {
        "102": "First name is required",

```

```

        "103": "Last name is required",
        "161": "Please enter your name",
        "201": "The file could not be sent.<br/><strong><p
class='filename' title='<%FilenameFull%>'>'<%FilenameTruncated%>'</p></strong><p
class='advice'>The maximum number of attached files would be exceeded
(<%MaxFilesAllowed%>)</p>",
        "202": "The file could not be sent.<br/><strong><p
class='filename' title='<%FilenameFull%>'>'<%FilenameTruncated%>'</p></strong><p
class='advice'>Upload limit and/or maximum number of attachments would be exceeded
(<%MaxAttachmentsSize%>)</p>",
        "203": "The file could not be sent.<br/><strong><p
class='filename' title='<%FilenameFull%>'>'<%FilenameTruncated%>'</p></strong><p
class='advice'>File type is not allowed.</p>",
        "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",
        "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.<br/><strong><p class='filename'
title='<%FilenameFull%>'>'<%FilenameTruncated%>'</p></strong><p class='advice'>File size is
larger than the allowed size (<%MaxSizePerFile%>)</p>",
        "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": "<div style='text-align:center'>Connection
lost</div>",
        "Reconnected": "<div style='text-align:center'>Connection
restored</div>",
        "FileSendFailed": "The file could not be sent.<br/><strong><p
class='filename' title='<%FilenameFull%>'>'<%FilenameTruncated%>'</p></strong><p
class='advice'>There was an unexpected disconnection. Try again?</p>",
        "Generic": "<div style='text-align:center'>An unexpected
error occurred</div>"
    }
}
}

```

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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.

Example

```
oMyPlugin.command('WebChat.configure', {  
  emojis: true,  
  actionsMenu: true,  
  inviteOnRestoreTimeout: true,  
  uploadsEnabled: true,  
  confirmCloseEnabled: true,  
  chatButton: {  
    enabled: false,  
    openDelay: 1000,  
    template: '<span>Done</span>',  
    effect: 'fade',  
    effectDuration: 1000,  
    hideDuringInvite: true  
  },  
  proactive: {  
    enabled: false,  
    idleTime: 1000,  
    cancelTimer: 1000  
  },  
});
```

```
        autoInvite:{
            enabled: true,
            timeToInviteSeconds: 1000,
            inviteTimeoutSeconds: 1000
        }
    }).done(function(e){
        // WebChat configured successfully
    }).fail(function(e){
        // WebChat wasn't configured properly
    });
```

Options

Option	Type	Description
emojis	boolean	Enable/disable Emoji menu inside chat message input.
actionsMenu	boolean	Enable/disable actions menu next to chat message input.
inviteOnRestoreTimeout	boolean	If set to true the invite will restore itself back on the screen after back end services are back online.
uploadsEnabled	boolean	If set to true you will be able to upload files during the chat session.
confirmFormCloseEnabled	boolean	If set to false confirm box will not show when closing chat box.
chatButton.enabled	boolean	Enable/disable chat button on screen.
chatButton.openDelay	number	Number of milliseconds before displaying chat button on screen.
chatButton.template	string	Custom HTML string template for chat button.
chatButton.effect	string	Type of animation effect when revealing chat button. 'slide' or 'fade'.
chatButton.effectDuration	number	Length of animation effect in milliseconds.
chatButton.hideDuringInvite	boolean	When auto-invite feature is activated, hide the chat button. When invite is dismissed, reveal the chat button again.
proactive.enabled	boolean	Enable/disable web engagement.
proactive.idleTimer	number	Watch the number of seconds of

Option	Type	Description
		idle time to invite a customer back.
proactive.cancelTimer	number	Number of seconds to cancel an invite from web engagement.
autoInvite.enabled	boolean	Enable/disable auto-invite feature. Automatically invites user to chat after user idles on page for preset time.
autoInvite.timeToInviteSeconds	number	Number of seconds of idle time before inviting customer to chat.
autoInvite.inviteTimeoutSeconds	number	Number of seconds to wait, after showing invite, before closing chat invite.

Resolutions

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

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'
  }
}).done(function(e){
  // WebChat opened successfully
}).fail(function(e){
```

```
    // WebChat isn't open or no active chat session  
});
```

Options

Option	Type	Description
userData	object	Object containing arbitrary data that gets sent to the server. Overrides userData set in the webchat configuration object.
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.

Resolutions

Status	When	Returns
resolved	When WebChat is successfully opened	n/a
rejected	When 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	When WebChat is successfully closed	n/a
rejected	When WebChat is already closed	'already closed'

minimize

Minimize or unminimize 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	When there is an active chat session to end	n/a
rejected	When there is no active chat session to end	'there is no active chat session to end'

invite

Show an invitation to chat using the Toaster popup element. Text shown in 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	When WebChat is closed and the toast element is created successfully	n/a
rejected	When 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 is unable to restore, this invitation will offer the user to start a new chat. Text shown in 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	When WebChat is closed, the config item 'webchat.inviteOnRestoreTimeout' is set, and the toast element is created successfully	n/a
rejected	When WebChat is already open. Prevents inviting a user that is already in a chat	'Chat is already open. Ignoring invite command.'

injectMessage

Inject 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: '<div>word</div>',  
      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	The border radius you'd like for

Option	Type	Description
	vale	the bubble.
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	When WebChat is open and there is an active chat session	An HTML reference (jQuery wrapped set) to the new injected message.
rejected	When WebChat is not open and/or there was no active chat session	'No chat session to inject into'

showChatButton

Makes the standalone chat button visible on the screen 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 show and hide animation.

Resolutions

Status	When	Returns
resolved	When the chat button is enabled in the configuration, is currently not visible, and the SideBar plugin is not initialized	n/a
rejected	When 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	When 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	When the chat button is currently visible	n/a
rejected	When the chat button is already hidden	'Chat button is already hidden. Ignoring command.'

showOverlay

A slide-down overlay the opens over WebChat's content. You can fill this overlay with content such as disclaimers, articles, and other information.

Example

```
oMyPlugin.command('WebChat.showOverlay', {
    html: '<div>Example text</div>',
    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

Option	Type	Description
		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	When WebChat is open and the overlay opens.	
rejected	When 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	When WebChat is open and the overlay closes.	
rejected	When WebChat is not currently open.	WebChat is not currently open. Ignoring command.

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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
closed	The WebChat widget has been removed from the screen	n/a
minimized	The WebChat widget has been changed to a minimized state	n/a
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 describe as 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
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, met with an agent, and the session ended normally:

```
ready -> opened -> started -> 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.
coBrowseInitiated	boolean	Indicates that a Co-browse session was started at some point during the chat session.
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.
opened	integer (timestamp)	Timestamp indicating when WebChat was opened.
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.
completed	integer (timestamp)	Timestamp indicating when the chat session ended normally. Completed refers to when a user

Name	Type	Description
		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.

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.

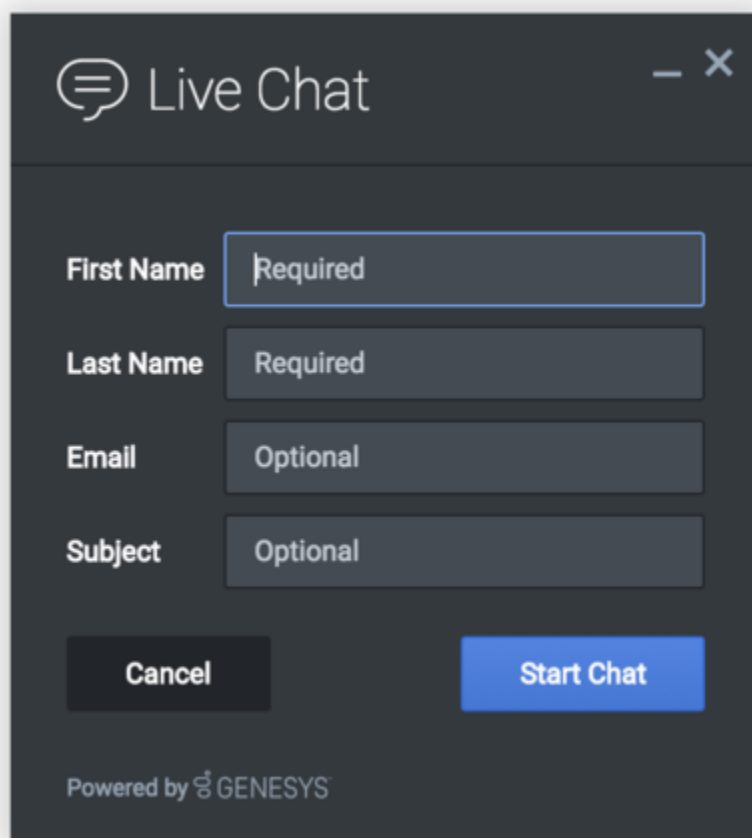
Default Example

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: "<table></table>",
  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"
    }
  ]
}
```

Using this JSON definition will result in this output:



The screenshot shows a 'Live Chat' widget interface. It features a dark-themed header with a chat icon and the text 'Live Chat'. Below the header, there are four input fields arranged vertically: 'First Name' (labeled 'Required'), 'Last Name' (labeled 'Required'), 'Email' (labeled 'Optional'), and 'Subject' (labeled 'Optional'). At the bottom of the form, there are two buttons: a dark 'Cancel' button and a blue 'Start Chat' button. The footer of the widget displays 'Powered by GENESYS' with a small logo.

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	"	
{label}	{input}		
<p>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.</p> <p>The default wrapper for an input is "</p>			
{label}	{input}		
validate	function		Define a validation

Property	Type	Default	Description
			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 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_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.

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

```
<input type="text" id="cx_webchat_form_firstname"
name="firstname" maxlength="100" placeholder="Required"></input>
```

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: "<table></table>", /* 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: "<tr><th>{label}</th><td>{input}</td></tr>" /* 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_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.

Validation Function Arguments

Argument	Type	Description
event	JavaScript event object	
form	HTML reference	A jquery reference to the form wrapper element.
input	HTML reference	A jquery reference to the input

Argument	Type	Description
		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.

SendMessageService

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

SendMessageService exposes a high-level API for utilizing Genesys send message services. You can use these services for sending a message to customer service on the front-end or for developing your own custom SendMessage widgets. Rather than developing a custom messaging UI and using the REST API directly, using SendMessageService drastically simplifies integration and greatly improves reliability, features, and compatibility on the bus for all widgets.

Usage

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

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

Namespace

SendMessage Service plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	sendmessage
CXBus - API Commands & API Events	SendMessageService

Customization

SendMessageService has no customization options. It is meant as a plug-n-play type of plugin and works as-is.

Configuration

Description

SendMessage and SendMessageService share the configuration namespace '_genesys.widgets.sendmessage'. SendMessage has UI options while SendMessageService has connection options.

Example

```
window._genesys.widgets.sendmessage = {  
  apiKey: 'n3eNkgLLgLKXREBYjGm6lyg0HH0K8VA',  
  dataURL: 'http://10.0.0.121:7777/genesys/2/email',  
  userData: {},  
  SendMessageButton: {  
    enabled: true,  
    template: '<div>Email</div>',  
    effect: 'fade',  
    openDelay: 1000,  
    effectDuration: 300,  
    hideDuringInvite: true  
  }  
};
```

Options

Name	Type	Description	Default	Required
apikey	string	Apigee Proxy secure token	n/a	Yes, if using Apigee Proxy
dataURL	URL String	URL of GMS server	n/a	Always
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout	3000	n/a
userData	object	Arbitrary JSON attached data to include with message	{}	n/a
uploadsEnabled	boolean	Enables file uploads to the server and enables the file upload	true	n/a

Name	Type	Description	Default	Required
		feature in the SendMessage UI plugin.		

Localization

No Localization options

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('SendMessageService.sendForm',{
    formData: {
        firstName: 'Bob',
        lastName: 'Jones',
        email: 'b.jones@mail.com',
        subject: 'product questions',
        text: 'Good morning',
        email: 'b.jones@mail.com'
    },
    userData: {},
    files: []
});
```

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('SendMessage.configure', {
    formValidation: true,
    SendMessageButton:{
        enabled: false,
        openDelay: 1000,
        template: '<span>Done</span>',
        effect: 'fade',
```



```
        effectDuration: 1000
    }
}).done(function(e){
    // SendMessage configured successfully
}).fail(function(e){
    // Invalid configuration
});
```

Options

Option	Type	Description
formValidation	boolean	Enable/disable browser form validations.
SendMessageButton.enabled	boolean	Enable/disable Send Message button on screen.
SendMessageButton.template	string	Custom HTML string template for Send Message button.
SendMessageButton.effect	string	Type of animation effect when revealing Send Message button ('slide' or 'fade').
SendMessageButton.openDelay	number	Number of milliseconds before displaying send message button on screen.
SendMessageButton.effectDuration	number	Length of animation effect in milliseconds.

Resolutions

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

sendForm

Sends a Message with the Email server via GMS. Intended to be used by Send Message widgets only. Should not be invoked manually.

Example

```
oMyPlugin.command('SendMessageService.sendForm', {
    formData: {
        firstName: 'Bob',
        lastName: 'Jones',
        email: 'b.jones@mail.com',
        subject: 'product questions',
        text: 'Good morning'
    },
    userData: {},
    files: []
}).done(function(e){
    // SendMessageService sent the form successfully
}).fail(function(e){
    // SendMessageService failed to send a form
});
```

Options

Option	Type	Description
formData.firstName	string	Send Message Entry Form Data: 'First Name'.
formData.lastName	string	Send Message Entry Form Data: 'Last Name'.
formData.email	string	Send Message Entry Form Data: 'Email'.
formData.subject	string	Send Message Entry Form Data: 'Subject'.
formData.text	string	Send Message Entry Form Data for message body content.
files	array	Array of file objects containing the attached files. Intended to be used by Send Message widgets only.
userData	object	Arbitrary data to attach to the message (AKA attachedData). Properties defined here will be merged with default userData set in the configuration object. If Genesys Web Engagement (GWE) is enabled, this userData also includes visitID, globalVisitID and pageID.

Resolutions

Status	When	Returns
resolved	When server confirms message sent	(AJAX Response Object)
rejected	When a browser does not support HTML5 form attachments	'No HTML5 formData support on your browser'
rejected	When no form data passed	'No formData found'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.


Important

The global bus object is a debug 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('SendMessageService.ready', function(e){});
```

Name	Description	Data
ready	SendMessageService is initialized and ready to accept commands	n/a
messageSent	Message is successfully sent	{interactionId: (interactionid)}
error	An error occurred between the client and the server	{result: (object), textStatus: (string), statusCode: (number), jqXHR: (string)}

SendMessage



Send Message

—×

First Name

Required

Last Name

Required


Email

Required


Subject

Required

Type your message here...

 **Attach files**

Send

Powered by  GENESYS

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

The SendMessage Widget provides a form for sending a message directly to customer service. Like an email, you can write a subject, body, and attach files. After sending your message an agent will respond back to the email provided.

Usage

SendMessage can be launched manually by the following methods:

- Calling the **command** "SendMessage.open"
- Configuring **ChannelSelector** to show SendMessage as a channel
- Enabling the built-in SendMessage launcher button that appears on the right side of the screen
- Create your own custom button or link to open SendMessage (using the "SendMessage.open" command)

Deployment Notes

SendMessage Service Configuration in GMS

In order to configure your SendMessage service in GMS, please follow these [instructions](#).

Customization

All text shown in the SendMessage Widget is fully customizable and **localizable** by adding entries into your **configuration** and **localization** options.

SendMessage supports themes. You may create and register your own themes for Genesys Widgets.

Namespace

Send Message plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	sendmessage
i18n - Localization	sendmessage
CXBus - API Commands & API Events	SendMessage
CSS	.cx-send-message

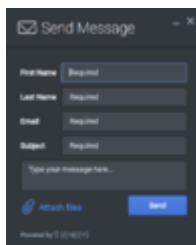
Mobile Support

SendMessage 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, SendMessage switches to special fullscreen templates that are optimized for both portrait and landscape orientations.

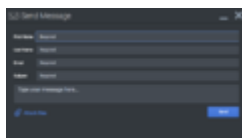
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

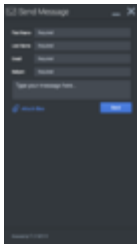
"Dark" Theme



Desktop docked view



Mobile fullscreen view in portrait orientation



Mobile fullscreen view in landscape orientation

"Light" Theme



Desktop docked view



Mobile fullscreen view in portrait orientation



Mobile fullscreen view in landscape orientation

Configuration

Description

SendMessage and SendMessageService share the configuration namespace '_genesys.widgets.sendmessage'. SendMessage has UI options while SendMessageService has connection options.

Example

```
window._genesys.widgets.sendmessage = {
  apiKey: 'n3eNkgLLgLKXREBYjGm6lyg0HH0K8VA',
  dataURL: 'http://10.0.0.121:7777/genesys/2/email',
  uploadsEnabled: true,

  SendMessageButton: {
    enabled: true,
    template: <div class="cx-icon" data-icon='email'></div>,
    effect: 'fade',
    openDelay: 1000,
    effectDuration: 300
  }
};
```

Options

Name	Type	Description	Default	Required
formValidation	boolean	Enable/Disable browser form validations.	true	n/a
SendMessageButton.enabled	boolean	Enable/Disable Send Message button on screen.	false	n/a
SendMessageButton.template	string	Custom HTML string template for Send Message button	<div class="cx-widget cx-send-message-button cx-side-button" data-message="SendMessageButton" data-gcb-service-	n/a

Name	Type	Description	Default	Required
			<pre>node="true"></div></pre>	
SendMessageButton. effect	effect	Type of animation effect when revealing Send Message button ('slide' or 'fade').	fade	n/a
SendMessageButton. openDelay	openDelay	Number of milliseconds before displaying send message button on screen.	1000	n/a
SendMessageButton. effectDuration	effectDuration	Length of animation effect in milliseconds.	300	n/a

Localization

Important

For information on how to setup localization, please refer to the [Localization Guide](#)

Usage

'sendmessage' namespace should be used when defining localization strings for Send Message plugin in your i18n JSON file.

In the example below, we demonstrate defining new strings for the 'en' (English) language. You may 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": {
    "sendmessage": {
      "title": "Send Message",
      "SendMessageButton": "Send Message",
      "EmailOk": "OK",
      "EmailFormPlaceholderFirstName": "Required",
      "EmailFormPlaceholderLastName": "Required",
      "EmailFormPlaceholderEmail": "Required",
      "EmailFormPlaceholderSubject": "Required",
      "EmailFormPlaceholderTypetextthere": "Type your message here...",
      "EmailFormFirstName": "First Name",
      "EmailFormLastName": "Last Name",
      "EmailFormEmail": "Email",
      "EmailFormSubject": "Subject",
      "EmailFormAttachfiles": "Attach files",
      "EmailFormSend": "Send",
      "ConfirmCloseWindow": "Are you sure you want to close the
<%GenesysWidget%> widget?",
      "SendMsgFormCancel": "Cancel",
      "SendMsgFormClose": "Close",
      "Errors": {
        "102": "First Name required",
        "103": "Last Name required",
        "104": "Subject required",
        "181": "Email address required",
        "182": "Message text content required",
        "connectionError": "Unable to reach server. Please try
```

```
again.",
    "unknownError": "Something went wrong, we apologize for the
inconvenience. Please check your connection settings and try again.",
    "attachmentsLimit": "Total number of attachments exceeds
limit: ",
    "attachmentsSize": "Total size of attachments exceeds limit:
",
    "invalidFileType": "Unsupported file type. Please upload
images, PDFs, text files and ZIPs.",
    "invalidFromEmail": "Invalid email - From address.",
    "invalidMailbox": "Invalid email - To address."
}
}
}
```

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('SendMessage.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.

Example

```
oMyPlugin.command('SendMessageService.configure', {  
    apiKey: '123456',  
    dataURL: 'http://localhost:8080/foo/mygms',  
    ajaxTimeout: 10000,  
    userData: {}  
}).done(function(e){  
    // SendMessageService configured successfully  
}).fail(function(e){  
    // SendMessageService failed to configure properly  
});
```

Options

Option	Type	Description
apikey	string	Apigee Proxy secure token
dataURL	URL String	URL of GMS server
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout
userData	object	Arbitrary JSON data to attach to the message.

Resolutions

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

open

opens the send message widget UI.

Example

```
oMyPlugin.command('SendMessage.open', {
  text: 'To whom it may concern.....',
  userData: {},
  form: {
    autoSubmit: false,
    firstname: 'John',
    lastname: 'Smith',
    email: 'John@mail.com',
    subject: 'Customer Satisfaction',
    messagebody: 'I am truly satisfied!'
  }
}).done(function(e){
  // SendMessage opens successfully
}).fail(function(e){
  // SendMessage isn't open or no active chat session
});
```

Options

Option	Type	Description
userData	object	Object containing arbitrary data that gets sent to the server. Overrides userData set in the sendmessage configuration object.
form	object	Object containing form data to prefill in the send message form and optionally auto-submit the form.
form.autoSubmit	boolean	Automatically submit the form and send an email with prefilled content.
form.validation	boolean	Enables/Disables validating the form data while submitting. By default, its enabled.
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.
text	string	value for the email body text content entry field

Resolutions

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

close

Closes the Send Message UI.

Example

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

Resolutions

Status	When	Returns
resolved	When Send Message is successfully closed	n/a
rejected	When Send Message is already closed	'already closed'

minimize

Minimize or Unminimize Send Message UI.

Example

```
oMyPlugin.command('SendMessage.minimize').done(function(e){  
    // SendMessage minimized successfully  
}).fail(function(e){  
    // SendMessage ignores command  
});
```

Resolutions

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

showSendMessageButton

Makes the standalone Send Message button visible on the screen using either the default template and CSS or customer-defined ones.

Example

```
oMyPlugin.command('SendMessage.showSendMessageButton', {  
    openDelay: 1000,  
    duration: 1500  
}).done(function(e){  
    // SendMessage shows send message button successfully  
}).fail(function(e){  
    // SendMessage button is already visible, side bar is active and overrides the send  
    message button, or chat button is disabled in configuration  
});
```

Options

Option	Type	Description
openDelay	number	Duration in milliseconds to delay showing the send message button on the page.
duration	number	Duration in milliseconds for the show and hide animation.

Resolutions

Status	When	Returns
resolved	When the Send Message button is enabled in the configuration, is currently not visible, and the SideBar plugin is not initialized	n/a
rejected	When the Send Message button is not enabled in the configuration, or it's already visible, or the SideBar plugin is initialized	'Send Message button is already visible. Ignoring command.'
rejected	When the SideBar plugin is active the standalone Send Message button will be disabled	'SideBar is active and overrides the default Send Message button'

Status	When	Returns
	automatically	

hideSendMessageButton

Hides the standalone Send Message button.

Example

```
oMyPlugin.command('SendMessage.hideSendMessageButton', {
    duration: 1000
}).done(function(e){
    // SendMessage shows send message button successfully
}).fail(function(e){
    // SendMessage button is already visible, side bar is active and overrides the send
    message button, or chat button is disabled in configuration
});
```

Options

Option	Type	Description
duration	number	Duration in milliseconds for the show and hide animation

Resolutions

Status	When	Returns
resolved	When the send message button is currently visible	n/a
rejected	When the send message button is already hidden	'Send Message button is already hidden. Ignoring command.'

submit

The user entered form data and attached files are submitted

Example

```
oMyPlugin.command('SendMessage.submit').done(function(e){  
    // SendMessage submitted form successfully  
}).fail(function(e){  
    // SendMessage failed to submit form  
});
```

Resolutions

Status	When	Returns
resolved	When Send Message form is submitted successfully	n/a
rejected	When form submit fails	'No form data found'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('SendMessage.ready', function(e){});
```

Name	Description	Data
opened	The Send Message widget has appeared on screen	n/a
ready	Send Message is initialized and ready to accept commands	n/a
minimized	The Send Message widget has been changed to a minimized state	n/a
unminimized	The Send Message widget has been restored from a minimized state to the standard view	n/a
closed	The Send Message widget has been removed from the screen	n/a

Metadata

Interaction Lifecycle

Every SendMessage 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 SendMessage), to the end (closing SendMessage), and every step in between.

The following events are part of the Interaction Lifecycle:

ready
opened
started
cancelled
completed
closed

Lifecycle Scenarios

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

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

ready -> opened -> cancelled -> closed

The user started filling out the form but closed SendMessage without sending it:

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

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

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

Tip

For a list of all SendMessage 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 `SendMessage` 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 email 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 <code>SendMessage</code> 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 <code>false</code> .
opened	integer (timestamp)	Timestamp indicating when <code>SendMessage</code> was opened.
started	integer (timestamp)	Timestamp indicating when the user started entering information into the form.
cancelled	integer (timestamp)	Timestamp indicating when the message draft is cancelled. Cancelled refers to when a user abandoned the interaction by closing <code>SendMessage</code> before sending a message.
completed	integer (timestamp)	Timestamp indicating when the message was sent successfully.
closed	integer (timestamp)	Timestamp indicating when <code>SendMessage</code> 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.

GWE

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

The GWE plugin allows for Genesys Widgets to interface with the Genesys Web Engagement product and services. GWE can invite users to start a chat session or send a message to customer service.

More information can be found on the [Genesys Web Engagement](#) product page.

Configuration

Description

Configuration for the Genesys Widgets GWE plugin are very basic, allowing you to set the secured or unsecured path to the GWE server. For a detailed list of configuration options for the GWE application, please visit the main documentation for GWE: [Genesys Web Engagement - Generating and Configuring the Instrumentation Script](#)

Example

```
window._genesys.widgets.gwe = {  
    httpEndpoint: 'http://www.website.com/gwe/',  
    httpsEndpoint: 'https://www.website.com/gwe/'  
};
```

Options

Name	Type	Description	Default	Required
httpEndpoint	string	URL/Path to the GWE server over standard HTTP	n/a	yes, if unsecured access available
httpsEndpoint	string	URL/Path to the GWE server over secure HTTPS	n/a	yes, if secured access available
smartInvites	boolean	When set to true, the smartInvites option will prevent proactive invites from appearing while the user is already using one of the widgets. This prevents redundancy and improves user experience. Disable to continue to show invites even if the user currently has a widget open.	true	No

Name	Type	Description	Default	Required
trackedEvents	object	An object list of Widgets events for GWE to track and send to the server. The object configured here will be blended with the default list of tracked events inside of the GWE plugin. You can disable the default events by specifying each one with a false value. You can add new events as well. for example <code>{'WebChat.opened': true}</code>	<pre>{ 'WebChatService.error': true, 'WebChatService.clientConnected': true, 'WebChatService.agentConnected': true, 'WebChatService.started': true, 'WebChatService.ended': true, 'WebChatService.disconnected': true, 'WebChat.opened': true, 'WebChat.closed': true, 'SendMessageService.error': true, 'SendMessage.opened': true, 'SendMessage.closed': true, 'CoBrowse.started': true, 'CoBrowse.stopped': true, 'CallbackService.scheduled': true, 'CallbackService.scheduleError': true, 'CallbackService.availabilityError': true, 'Callback.opened': true, 'Callback.closed': true, 'CallUs.opened': true, 'CallUs.closed': true, 'ChannelSelector.opened': true, 'ChannelSelector.closed': true }</pre>	No

Localization

No localization options

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('GWE.getIds');
```

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('GWE.configure', {  
    httpEndpoint: 'http://localhost:8080/foo/bar',  
    httpsEndpoint: 'https://localhost:8080/foo/bart'  
}).done(function(e){  
    // GWE configured successfully  
}).fail(function(e){  
    // GWE wasn't configured properly  
});
```

Options

Option	Type	Description
httpEndpoint	string	URL/Path to the GWE server over standard HTTP
httpsEndpoint	string	URL/Path to the GWE server over secure HTTPS

Resolutions

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

getIDs

Return Ids of Web Engagement items

Example

```
oMyPlugin.command('GWE.getIDs').done(function(e){  
    // GWE got IDs successfully  
}).fail(function(e){  
    // GWE did not get IDs  
});
```

Resolutions

Status	When	Returns
resolved	When IDs are valid or if they are available.	Array of IDs or nothing

invite

Show an invitation using the Toaster popup element.

Example

```
oMyPlugin.command('GWE.invite', {
    container: {},
    content: 'content of view'
}).done(function(e){
    // GWE showed invite successfully
}).fail(function(e){
    // GWE failed to show invite properly
});
```

Options

Option	Type	Description
container	object	Container object
content	string	Content within the web engagement view

Resolutions

Status	When	Returns
resolved	When web engagement information properly provided	n/a

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('GWE.ready', function(e){});
```

Name	Description	Data
ready	The GWE plugin is initialized and ready on the bus	n/a

CoBrowse

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

The CoBrowse plugin allows for Genesys Widgets to interface with the Genesys Co-browse product and services to start and stop Co-browse sessions.

More information can be found on the [Genesys Co-browse](#) product page.

Deployment Notes

Blocking Pages from Agent View

Having the co-browse instrumentation removed from certain pages allows customers to block those pages from the agent view completely. In order to block certain pages, use one of the following methods:

Method 1: Do not configure co-browse on that page. CX Widgets only initializes and injects co-browse instrumentation when co-browse configuration is set.

Method 2: Override loaded plugins using configuration:

Examples

Normal pages

```
_genesys.widgets.main.plugins = ["cx-webchat", "cx-webchat-service", "cx-cobrowse"]
```

Bypass Cobrowse

```
_genesys.widgets.main.plugins = ["cx-webchat", "cx-webchat-service"]
```


Configuration

Description

The CoBrowse plugin has both configuration options for Genesys Widgets and configuration options for the Co-browse application itself. Listed on this page are the configuration options for the Genesys Widgets CoBrowse plugin which are defined in the global configuration object here:

'_genesys.cxwidget.cobrowse'. Configuration objects for the Co-browse application can be set in either 'window._genesys.cobrowse' or 'window._genesys.cxwidget.cobrowse'. For a detailed list of configuration options for the Co-browse application, please visit the main documentation for Co-browse: <https://docs.genesys.com/Documentation/GCB>

Example

```
window._genesys.widgets.cobrowse = {  
    src: 'https://www.website.com/cobrowse/js/gcb.min.js',  
    url: 'https://www.website.com/cobrowse/'  
};
```

Options

Name	Type	Description	Default	Required
src	string	URL/Path to the Co-browse JavaScript package. Usually resides on the Co-browse server.	n/a	Always
url	string	URL/Path to the Co-browse server endpoint	n/a	Always

Localization

Please see localization options in the main [Co-browse documentation](#).

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('CoBrowse.start');
```

start

Start a Co-browse session

Example

```
oMyPlugin.command('CoBrowse.start').done(function(e){  
    // Co-browse started a session successfully  
}).fail(function(e){  
    // Co-browse failed to start a session  
});
```

Resolutions

Status	When	Returns
resolved	Co-browse API is available and used to start session	n/a
rejected	Co-browser API is not available	n/a

stop

Stop the currently active Co-browse session

Example

```
oMyPlugin.command('CoBrowse.stop').done(function(e){
    // Co-browse stopped a session successfully
}).fail(function(e){
    // Co-browse failed to stop a session
});
```

Resolutions

Status	When	Returns
resolved	Co-browse API is available and used to end the active session	n/a
rejected	Co-browser API is not available	n/a

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('CoBrowse.configure', {
    src: 'http://localhost:8080/foo/sample',
    url: 'http://localhost:8080/foo/bar'
}).done(function(e){
    // Co-browse configured successfully
}).fail(function(e){
    // Co-browse wasn't configured properly
});
```

Options

Option	Type	Description
src	string	URL/Path to the Co-browse JavaScript package. Usually resides on the Co-browse server.
url	string	URL/Path to the Co-browse server endpoint

Resolutions

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

open

Opens the Co-browse UI.

Example

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

Resolutions

Status	When	Returns
resolved	When Co-browse is successfully opened	n/a
rejected	When Co-browse is already open	'Already opened'

close

Closes the Co-browse UI.

Example

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

Resolutions

Status	When	Returns
resolved	When Co-browse successfully closed	n/a
rejected	When Co-browse is already closed	'Already closed'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('CoBrowse.ready', function(e){});
```

Name	Description	Data
started	A Co-browse session has been started	n/a
stopped	An active Co-browse session has been stopped	n/a
online	Additional JS files have been loaded and a connection to the server has been established	n/a
ready	The CoBrowse plugin is initialized and ready on the bus	n/a

App

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

App is the main controller for Genesys Widgets and has no UI. It controls all startup routines, global configurations, Extensions, 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,
  timeFormat: 12
  plugins: [
    'cx-channel-selector',
    'cx-webchat-service',
    'cx-webchat',
    'cx-call-us',
    'cx-cobrowse',
    'cx-send-message',
    'cx-send-message-service',
    'cx-gwe',
    'cx-stats-service'
  ],
  debug: true
}
```

Options

Name	Type	Description	Default	Required
themes	object	An object list containing the CSS classname for each Theme. The	{dark: 'cx-theme-dark', light: 'cx-theme-light'}	

Name	Type	Description	Default	Required
		<p>property names are used to select the theme in the 'theme' property. for example</p> <pre>{ dark:'cx-theme-dark', light:'cx-theme-light', 'red':'cx-theme-red', 'blue':'cx-theme-blue'}</pre> <p>Where 'dark' and 'light' are the built-in themes provided in Genesys Widgets and 'red' and 'blue' are example custom theme names you may create on your own. Note: 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>		
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	
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	en	

Name	Type	Description	Default	Required
		more information about localization, see localization .		
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 .	Default English language strings are built into each widget and are displayed by default. Defining this i18n language pack overrides the built-in strings.	
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	
mobileModeBreakpoint	number	The breakpoint width in pixels where Genesys Widgets will switch to Mobile Mode. Breakpoint checked at startup only.	600	
timeFormat	number/string	This sets the time format for the timestamps. It can be 12 or 24	12	
plugins	array	A list of plugin names. This list overrides the default list of plugins that are initialized at startup. You can use this array to dynamically change the available plugins so that different user types have access to different plugins.	['cx-channel-selector', 'cx-webchat-service', 'cx-webchat', 'cx-call-us', 'cx-cobrowse', 'cx-send-message', 'cx-send-message-service', 'cx-gwe', 'cx-stats-service']	

Name	Type	Description	Default	Required
debug	boolean	Enable debug logging from the bus to appear in the browser console.	false	
customStylesheetID	string	The HTML ID of a <style> tag that contains CSS overrides, custom themes, or other custom CSS intended for Genesys Widgets. On startup, Widgets will move this <style> tag to the end of the document so that 1:1 CSS class overrides apply correctly.		

Localization

No localization options.

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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 <code>window._genesys.main.themes</code> . 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 'light'
rejected		'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 'cx-theme-light'
rejected	Never	

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 new classname applied.

Example

```
oMyPlugin.command('App.reTheme', {html: '<div>Test Theme</div>'}).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	When HTML is provided and theme is updated	HTML that was passed-in and modified
rejected	When 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 2000 milliseconds.

Resolutions

Status	When	Returns
resolved	Always	
rejected	Never	

setLanguage

Important

Internal use only.

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	When language successfully changed	
rejected	When no language code is provided	No language code provided
rejected	When 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 closed all successfully  
}).fail(function(e){  
    // App failed to close all  
});
```

Resolutions

Status	When	Returns
resolved	Always	
rejected	Never	

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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 being set for the first time.	{theme: '(theme CSS classname)'}
timeFormat	Published when the time format for Genesys Widgets is changed or is being set for the first time.	{timeFormat: iTimeFormat}

Calendar

The screenshot shows a 'Receive a Call' widget interface. On the left, there are input fields for 'First Name' (Required), 'Last Name' (Required), 'Phone' (+1), and 'Email' (Optional). Below these is a 'Date & Time' section with a radio button for 'As Soon As Possible'. A calendar overlay is displayed in the center, showing a grid of time slots for the next four days: Today (25 Aug), Tomorrow (26 Aug), Sunday (27 Aug), and Monday (28 Aug). The 'Morning' section is selected, and the time slot '09:25 AM' on Sunday is highlighted. At the bottom of the overlay, a radio button is selected for 'Today - 25 Aug - 09:25 AM'. The interface includes 'Cancel' and 'Confirm' buttons at the bottom, and a 'Powered by GENESYS' logo at the bottom left.

	Today 25 Aug	Morning	Afternoon	Evening	
Tomorrow 26 Aug		09:00 AM	09:05 AM	09:10 AM	09:15 AM
Sunday 27 Aug		09:20 AM	09:25 AM	09:30 AM	09:35 AM
Monday 28 Aug		09:40 AM	09:45 AM	09:50 AM	09:55 AM
Tuesday 29 Aug		10:00 AM	10:05 AM	10:10 AM	10:15 AM

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

Calendar widget is a UI Plugin that displays time-slots for a selected day. Each day is divided into 3 sections (for example, 'Morning', 'Afternoon' and 'Evening'). The number of days to display, as well as open time and close time for a day are configurable as shown in [Configuration](#).

Usage

Important

By default the calendar widget needs a UI container to display itself properly. Please refer to the [calendar generated events](#) to get the calendar and to display it where you prefer.

- Enable/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](#) configuration to enable only those time-slots for which a customer service agent is available and disable the remaining.
- Define your own [time interval](#) between each time-slot.

Customization

All the texts shown in calendar widget are fully localizable as shown in [Localization](#)

Namespace

Calendar plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	calendar
i18n - Localization	calendar
CXBus - API Commands & API Events	Calendar
CSS	.cx-calendar

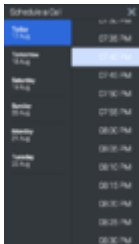
Mobile Support

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

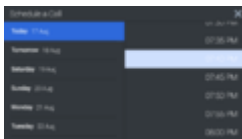
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

"Dark" Theme

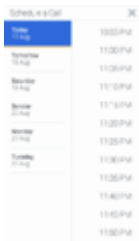


Mobile fullscreen view in portrait orientation



Mobile fullscreen view in landscape orientation

"Light" Theme



Mobile fullscreen view in portrait orientation



Mobile fullscreen view in landscape orientation

Configuration

Description

Calendar share the configuration namespace '_genesys.widgets.calendar'. Calendar has UI options.

Example

```
window._genesys.widgets.calendar = {  
  showAvailability: true,  
  numberOfDays: 5,  
  timeFormat: 12  
  calendarHours: {  
    interval: 10,  
    morning: {  
      enable: true,  
      openTime: '09:00',  
      closeTime: '11:59'  
    },  
    afternoon: {  
      enable: true,  
      openTime: '12:00',  
      closeTime: '16:59'  
    },  
    evening: {  
      enable: true,  
      openTime: '17:00',  
      closeTime: '23:59'  
    }  
  }  
};
```

Options

Name	Type	Description	Default	Required
timeFormat	number/string	This sets the time format for the timestamps in this widget. It can be	12	

Name	Type	Description	Default	Required
		12 or 24		
showAvailability	boolean	Enable/disable calendar to update the timeslots based on the callback availability. The unavailable timeslots are greyed out.	true	n/a
numberOfDays	number	The number of days to display on calendar starting today.	5	n/a
calenderHours.interval	number	The time interval between each consecutive timeslot displayed on calendar.	15	n/a
calenderHours.morning	boolean	Enable/Disable morning section in calendar.	true	n/a
calenderHours.morningOpenTime	number	Morning section opening time in 'HH:MM' 24 Hr format.	08:00	n/a
calenderHours.morningCloseTime	number	Morning section closing time in 'HH:MM' 24 Hr format.	11:59	n/a
calenderHours.afternoon	boolean	Enable/Disable afternoon section in calendar.	true	n/a
calenderHours.afternoonOpenTime	number	Afternoon section opening time in 'HH:MM' 24 Hr format.	12:00	n/a
calenderHours.afternoonCloseTime	number	Afternoon section closing time in 'HH:MM' 24 Hr format.	16:59	n/a
calenderHours.evening	boolean	Enable/Disable evening section in calendar.	true	n/a
calenderHours.eveningOpenTime	number	Evening section opening time in 'HH:MM' 24 Hr format.	17:00	n/a
calenderHours.eveningCloseTime	number	Evening section closing time in	23:59	n/a

Name	Type	Description	Default	Required
		'HH:MM' 24 Hr format.		

Localization

Important

For information on how to setup localization, please refer to the [Localization Guide](#)

Usage

'calendar' namespace should be used when defining localization strings for Calendar plugin in your i18n JSON file.

In the example below, we demonstrate defining new strings for the 'en' (English) language. You may 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": {
    "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",
    }
  }
}
```

```
        "CalendarLabelMorning": "Morning",
        "CalendarLabelAfternoon": "Afternoon",
        "CalendarLabelEvening": "Evening",
        "CalendarTitle": "Schedule a Call",
        "CalendarOkButtonText": "Okay",
        "CalendarError": "Unable to fetch availability details."
    }
}
```

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('Calendar.reset');
```

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('Calendar.configure', {  
  showAvailability: true,  
  numberOfDays: 7,  
  calendarHours:{  
    interval: 3000,  
    morning:{  
      enable: true,  
      openTime: '08:00',  
      closeTime: '11:00'  
    },  
    afternoon:{  
      enable: true,  
      openTime: '11:01',  
      closeTime: '14:30'  
    },  
    evening:{  
      enable: true,  

```

```

        openTime: '14:31',
        closeTime: '18:00'
    },
    }
  }).done(function(e){
    // Calendar configured successfully
  }).fail(function(e){
    // Calendar failed because of invalid configuration or invalid time is provided
  });

```

Options

Option	Type	Description
showAvailability	boolean	Enable/disable calendar to update the timeslots based on the callback availability. The unavailable timeslots are greyed out.
numberOfDays	number	The number of days to display on calendar starting today.
calenderHours.interval	number	The time interval between each consecutive timeslot displayed on calendar.
calenderHours.morning.enable	boolean	Enable/Disable morning section in calendar.
calenderHours.morning.openTime	string	Morning section opening time in 'HH:MM' 24 Hr format.
calenderHours.morning.closeTime	string	Morning section closing time in 'HH:MM' 24 Hr format.
calenderHours.afternoon.enable	boolean	Enable/Disable afternoon section in calendar.
calenderHours.afternoon.openTime	string	Afternoon section opening time in 'HH:MM' 24 Hr format.
calenderHours.afternoon.closeTime	string	Afternoon section closing time in 'HH:MM' 24 Hr format.
calenderHours.evening.enable	boolean	Enable/Disable evening section in calendar.
calenderHours.evening.openTime	string	Evening section opening time in 'HH:MM' 24 Hr format.
calenderHours.evening.closeTime	string	Evening section closing time in 'HH:MM' 24 Hr format.

Resolutions

Status	When	Returns
resolved	When configuration options are provided and set	n/a
rejected	When no configuration options are provided	'No configuration provided'
rejected	When invalid time is provided in Morning opening time section	'Invalid config: morning open time'
rejected	When invalid time is provided in Morning closing time section	'Invalid config: morning close time'
rejected	When invalid time is provided in Afternoon opening time section	'Invalid config: afternoon open time'
rejected	When invalid time is provided in Afternoon closing time section	'Invalid config: afternoon close time'
rejected	When invalid time is provided in Evening opening time section	'Invalid config: evening open time'
rejected	When invalid time is provided in Evening closing time section	'Invalid config: evening close time'

generate

Builds and generates the calendar. Should subscribe to the events to get the generated calendar and display 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 timeslots with the callback availability. This enables only those timeslots that have the callback facility and disables the remaining.

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 callback availability on calendar with the selected date.

Resolutions

Status	When	Returns
resolved	When timeslots are successfully updated	n/a
rejected	When no date value is found to check the availability	'No date found to check 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 own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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: <Generated HTML Calendar>, refresh!Scroll: <callback function to refresh scroll on Calendar>, destroy!scroll: <callback function to destroy scroll instance on Calendar> }
selectedDateTime	Date and time selected on calendar	{ dayString: <The day selected on calendar>, dateString: <The date selected on calendar in DD MMM format>, timeString: <The time selected on calendar in HH:MM 12 Hr format>, date: <Entire Date in date string format> }

CallbackService

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

CallbackService exposes a high-level API for utilizing Genesys Callback services. You can use these services to schedule a callback with customer service using our callback widget or by developing your own custom Callback widget. Using CallbackService drastically simplifies integration and greatly improves reliability, features, and compatibility on the bus for all widgets.

Usage

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

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

Namespace

Callback Service plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	sendmessage
CXBus - API Commands & API Events	CallbackService

Customization

CallbackService has no customization options. It is meant as a plug-n-play type of plugin and works as-is.

Configuration

Description

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

Example

```
window._genesys.widgets.callback = {  
  apiKey: 'n3eNkgLLgLKXREBYjGm6lygOHH0K8VA',  
  dataURL: 'http://fce-w0147.us.int.genesyslab.com:8010/genesys/1/service/callback/  
samples',  
  userData: {},  
  countryCodes: true,  
  formValidation: false,  
};
```

Options

Name	Type	Description	Default	Required
apikey	string	Apigee Proxy secure token	n/a	Yes, if using Apigee Proxy
dataURL	URL String	URL of GMS callback server	n/a	Always
userData	object	Arbitrary JSON attached data to include while scheduling a callback	{}	
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout	3000	

Localization

No Localization options

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('CallbackService.schedule', {  
    userData: {},  
    firstname: 'Bob',  
    lastname: 'Jones',  
    email: 'b.jones@mail.com',  
    subject: 'product questions',  
    desiredTime: '2017-04-04T00:24:17.804Z',  
    email: 'b.jones@mail.com',  
    phonenumber: '4151110000'  
});
```

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('CallbackService.configure', {  
    apikey: '123456',  
    dataURL: 'http://localhost:8080/foo/bar',  
    ajaxTimeout: 10000,  
    userData: {}  
}).done(function(e){  
    // CallbackService configured successfully  
}).fail(function(e){
```

```
}); // CallbackService failed to configure properly
```

Options

Option	Type	Description
apikey	string	Apigee Proxy secure token
dataURL	URL String	URL of GMS callback server
userData	object	Arbitrary JSON attached data to include when scheduling a callback
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout

Resolutions

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

schedule

Schedule a callback service with the GMS callback schedule API.

Example

```
oMyPlugin.command('CallbackService.schedule', {  
  userData: {},  
  firstname: 'Bob',  
  lastname: 'Jones',  
  email: 'b.jones@mail.com',  
  subject: 'product questions',  
  desiredTime: '2017-03-03T00:24:17.804Z',  
  email: 'b.jones@mail.com',  
  phonenummer: '4151110000'  
}).done(function(e){  
  // CallbackService scheduled successfully
```

```
}).fail(function(e){  
    // CallbackService failed to schedule  
});
```

Options

Option	Type	Description
firstname	string	Receive a Call entry Form Data: 'firstname'.
lastname	string	Receive a Call entry Form Data: 'lastname'.
phonenummer	string	Receive a Call entry Form Data: 'phonenummer'.
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. If Genesys Web Engagement (GWE) is enabled, this userData also includes visitID, globalVisitID and pageID.

Resolutions

Status	When	Returns
resolved	When server confirms callback is scheduled.	(AJAX Response Object)
rejected	When selected timeslot is not available.	(AJAX Response Object)
rejected	When AJAX exception occurs.	(AJAX Response Object)
rejected	When server exception occurs.	(AJAX Response Object)
rejected	When no form data is found to schedule callback.	'No data found to schedule callback'

availability

Get the list of available callback timeslots via GMS callback service.

Example

```
oMyPlugin.command('CallbackService.availability', {
    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.

Resolutions

Status	When	Returns
resolved	When server confirms the list of available callback timeslots.	(AJAX Response Object)
rejected	When timeslots are available for selected period.	(AJAX Response Object)
rejected	When AJAX exception occurs.	(AJAX Response Object)
rejected	When server exception occurs.	(AJAX Response Object)
rejected	When no query data is found.	'No query parameters passed for callback availability service'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('CallbackService.ready', function(e){});
```

Name	Description	Data
ready	CallbackService is initialized and ready to accept commands	n/a
scheduled	Callback is scheduled successfully	{{success data}}
scheduleError	An error occurred between the client and the server during a callback schedule	{result: (object), textStatus: (string), statusCode: (number), message: (string)}
availableSlots	Callback available slots fetched successfully	{{success data}}
availabilityError	An error occurred between the client and the server while fetching the available timeslots	{result: (object), textStatus: (string), statusCode: (number)}

Callback

Receive a Call

Please fill out the following details. Today, 15 Aug at 13:58

First Name Required

Last Name Required

Phone +1

Email Optional

Date & Time When should we call you? (Required)

☒ As Soon As Possible ☐ Custom Date & Time

Cancel **Confirm**

Powered by GENESYS

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

The Callback Widget provides a form to fetch user details like Name, Phone number, Email and Callback Immediate / Call at a user preferred time and submits to the Customer Service. Preferred times are updated based on the agents availability so that user can pick one.

Usage

Callback can be launched manually by the following methods:

- Calling the **command** "Callback.open"
- Configuring **ChannelSelector** to show Receive a Call as a channel
- Configuring **Calendar** to show a Date-Time picker for selecting a preferred time.

Dependency

The Callback Widget needs **Calendar** plugin. Make sure that it is included.

Customization

All text shown in the Callback Widget is fully customizable and **localizable** by adding entries into your **configuration** and **localization** options.

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

Namespace

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 fullscreen templates that are optimized for both portrait and landscape orientations.

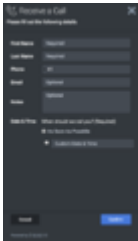
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

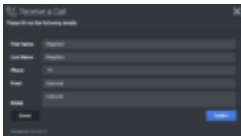
"Dark" Theme



Callback with Calendar in desktop



Mobile fullscreen view in portrait orientation



Mobile fullscreen view in landscape orientation

"Light" Theme



Desktop Callback view



Mobile fullscreen view in portrait orientation



Mobile fullscreen view showing country codes for phone numbers

Configuration

Description

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

Example

```
window._genesys.widgets.callback = {  
  apiKey: 'n3eNkgLLgLKXREBMjGm6lygOHH0K8VA',  
  dataURL: 'http://fce-w0147.us.int.genesyslab.com:8010/genesys/1/service/callback/  
samples',  
  userData: {},  
  countryCodes: true,  
  formValidation: false,  
  immediateCallback: true,  
  scheduledCallback: true,  
  ewt: {  
    queue: 'chat_ewt_test',  
    immediateCallback: {  
      thresholdMin: 10,  
      thresholdMax: 3000  
    }  
  }  
};
```

Options

Name	Type	Description	Default	Required
countryCodes	boolean	Enable/disable country flags display for phone numbers.	true	n/a
immediateCallback	boolean	Enable/disable the 'As Soon As Possible' option. Set value to true to enable.	true	n/a
scheduledCallback	boolean	Enable/disable the scheduling option. Set value to true to enable.	true	n/a

Name	Type	Description	Default	Required
ewt.queue	string	EWT service channel virtual queue.	none	always required if the range for EWT is configured.
ewt.immediateCallbackThresholdMin	string	If EWT is less than this minimum threshold value (seconds), the 'As Soon As Possible' option will be disabled.	none	none
ewt.immediateCallbackThresholdMax	string	If EWT is more than this maximum threshold value (seconds), the 'As Soon As Possible' option will be disabled.	none	none
formValidation	boolean	Enable/Disable form validations. These include validating Required fields, Invalid email address and Phone number. The type of validation applied on phone number is to check if its minimum length is 9 (including country code format).	true	n/a

Localization

Important

For information on how to setup localization, please refer to the [Localization Guide](#)

Usage

'callback' namespace should be used when defining localization strings for Callback plugin in your i18n JSON file.

In the example below, we demonstrate defining new strings for the 'en' (English) language. You may 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",
      "CallbackTitleDescription": "Please fill out the following details.",
      "CancelButtonText": "Cancel",
      "ConfirmButtonText": "Confirm",
      "CallbackFirstName": "First Name",
      "CallbackPlaceholderFirstName": "Required",
      "CallbackLastName": "Last Name",
      "CallbackPlaceholderLastName": "Required",
      "CallbackPhoneNumber": "Phone",
      "CallbackPlaceholderPhoneNumber": "Required",
      "CallbackEmail": "Email",
      "CallbackPlaceholderEmail": "Optional",
      "CallbackPlaceholderNotes": "Optional",
      "CallbackDateTime": "Date & Time",
      "CallbackDateTimeDescription": "When should we call you? (Required)",
      "CallbackTodayDate": "Today",
      "CallbackDayLabels": [
        "Sunday",
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday"
      ],
    },
  },
}
```

```

        "CallbackMonthLabels": [
            "Jan",
            "Feb",
            "Mar",
            "Apr",
            "May",
            "Jun",
            "Jul",
            "Aug",
            "Sep",
            "Oct",
            "Nov",
            "Dec"
        ],
        "CallbackTimeAtText": "at",
        "CallbackRadioButtonText": "As Soon As Possible",
        "CallbackBookedPhoneNumberLabel": "Phone",
        "CallbackBookedDateTimeLabel": "Date & Time",
        "CallbackBookedNotesLabel": "Optional Notes",
        "CallbackBookedDescription": "You're booked in!",
        "CallbackNotes": "Notes",
        "CallbackDone": "Done",
        "CallbackOk": "Okay",
        "CallbackCloseConfirm": "Are you sure you want to cancel arranging
this callback?",
        "CallbackNoButtonText": "No",
        "CallbackYesButtonText": "Yes",
        "CallbackPlaceholderCalendar": "Custom Date & Time",
        "Errors": {
            "unknownError": "Something went wrong, we apologize for the
inconvenience. Please check your connection settings and try again."
        }
    }
}

```

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('Callback.open');
```

open

Opens the Callback UI.

Example

```
oMyPlugin.command('Callback.open', {  
  form: {  
    autoSubmit: false,  
    firstname: 'John',  
    lastname: 'Smith',  
    email: 'John@mail.com',  
    subject: 'Customer Satisfaction',  
    desiredTime: 'now',  
    phonenumber: '8881110000'  
  }  
}).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.email	string	Value for the email entry field.
form.subject	string	Value for the notes entry field.
form.desiredTime	string	This Value is shared by the Date & Time section in the callback form. A string value 'now' pre-selects 'As soon As Possible' section and sets current local time as the desired time. When a string Date Time value/date object is passed, the desired time is set to this value in the Custom Date & Time input field. During form submission to the server desired time time is converted into UTC string format.
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.
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	When callback form is successfully opened	n/a
rejected	When callback form is already open	'already opened'

close

Closes the Callback UI.

Example

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

Resolutions

Status	When	Returns
resolved	When Callback form is successfully closed	n/a
rejected	When Callback form 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('Callback.configure', {  
    formValidation: true,  
    countryCodes: true  
}).done(function(e){  
    // Callback configured successfully  
}).fail(function(e){  
    // Callback wasn't configured properly  
});
```

Options

Option	Type	Description
formValidation	boolean	Enable/disable browser form validations.
countryCodes	boolean	Enable/disable country flags display for phone numbers.

Resolutions

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

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('Callback.ready', function(e){});
```

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

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  
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 -> 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

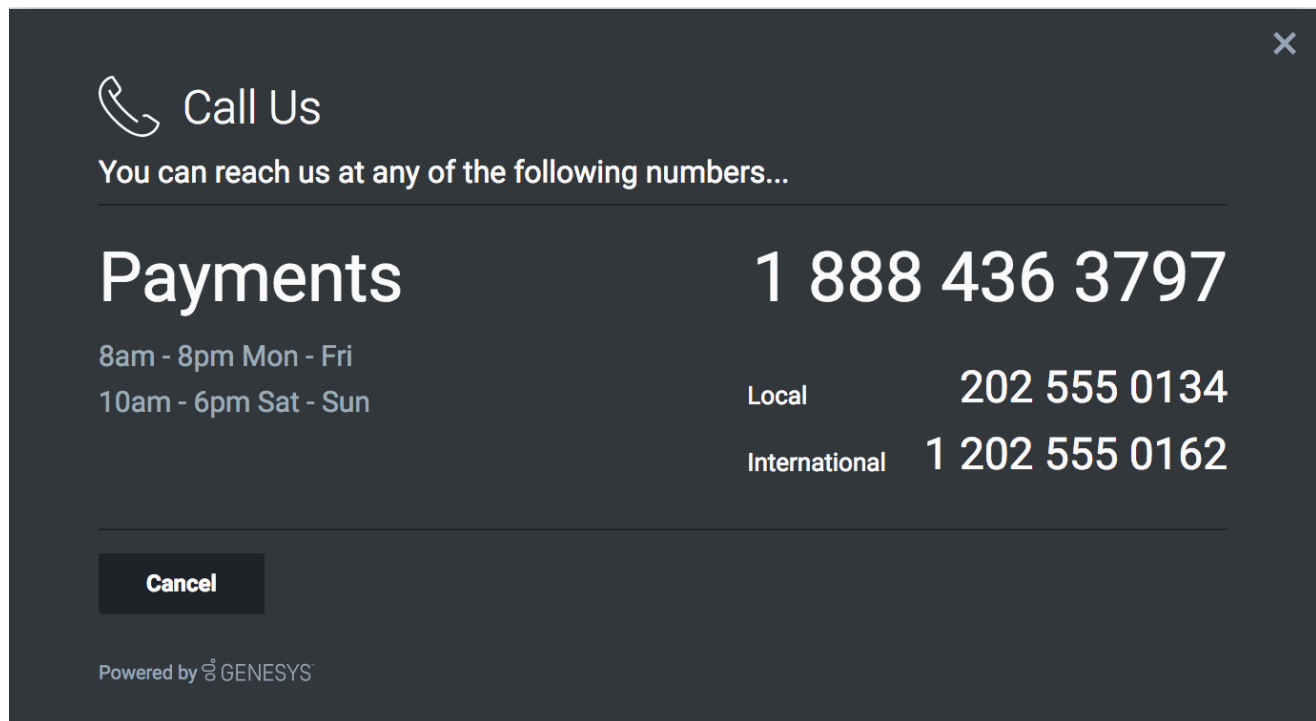
will be updated.

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.
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.

CallUs



- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

The CallUs Widget provides an overlay screen showing one or more phone numbers for a 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 followed by optional alternative or additional phone numbers. Each can be named and there is no limit on the amount of phone numbers you can include. If the list of numbers cannot fit in the widget, the user can scroll to see the remaining numbers.

Usage

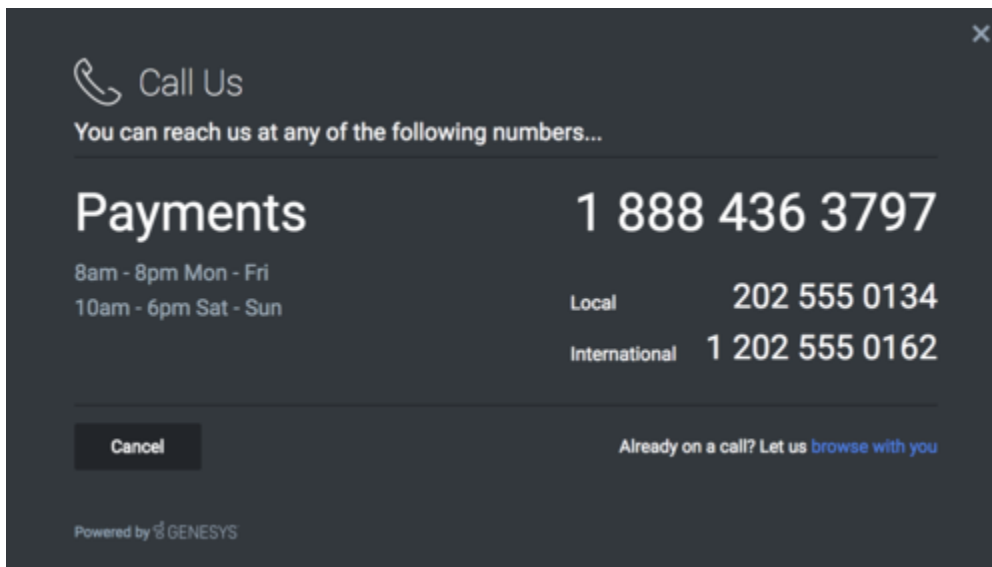
CallUs can be launched manually by the following methods:

- Calling the **command** "CallUs.open"
- Configuring **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.

Co-browse link



Desktop overlay view with Co-browse

Co-browse is integrated into CallUs and can be indicated on the bottom right of the CallUs Widget. CallUs will detect if Co-browse is available based on your configuration. If Co-browse is available the link will be visible, if not the link will not be visible.

Customization

All text, titles, names and numbers shown in the CallUs Widget are fully customizable and **localizable** 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 may create and register your own themes for Genesys Widgets.

Namespace

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

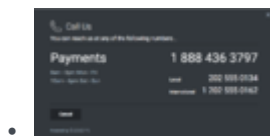
Mobile Support

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 fullscreen 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



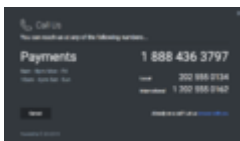
Desktop overlay view



Mobile fullscreen view in portrait orientation



Mobile fullscreen view in landscape orientation



Desktop overlay view with Co-browse



Mobile fullscreen view with Co-browse in portrait orientation



Mobile fullscreen view with Co-browse in landscape orientation

"Light" theme



Desktop overlay view



Mobile fullscreen view in portrait orientation



Mobile fullscreen view in landscape orientation



Desktop overlay view with Co-browse



Mobile fullscreen view with Co-browse in portrait orientation



Mobile fullscreen view with Co-browse in landscape orientation

Configuration

Description

CallUs uses the configuration property '_genesys.widgets.callus'. You must specify all 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	An array of objects that represent phone numbers and their labels. The first number in this list will display as the larger, main	[]	true

Name	Type	Description	Default	Required
		<p>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>Ex:</p> <pre>{ "displayName": "Payments", "i18n": "Number001", "number": "1 202 555 0162" }</pre>		
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 setup localization, please refer to the [Localization Guide](#)

Usage

'callus' namespace should be used when defining localization strings for CallUs plugin in your i18n JSON file.

In the example below, we demonstrate defining new strings for the 'en' (English) language. You may 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",
      "CoBrowseText": "<span class='cx-cobrowse-offer'>Already on a call?
Let us <a href='javascript:void(0);' class='cx-cobrowse-link'>browse with you</a></span>",
      "CoBrowse": "Start Co-browse",
      "CoBrowseWarning": "Co-browse allows your agent to see and control
your desktop, to help guide you. When on a live call with an Agent, request a code to start
Co-browse and enter it below. Not yet on a call? Just cancel out of this screen to return to
Call Us page."
    }
  }
}
```

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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	When CallUs is successfully opened	n/a
rejected	When 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	When CallUs successfully closed	n/a
rejected	When 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	When CallUs configuration is provided	n/a
rejected	When no configuration provided	'Invalid Configuration'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

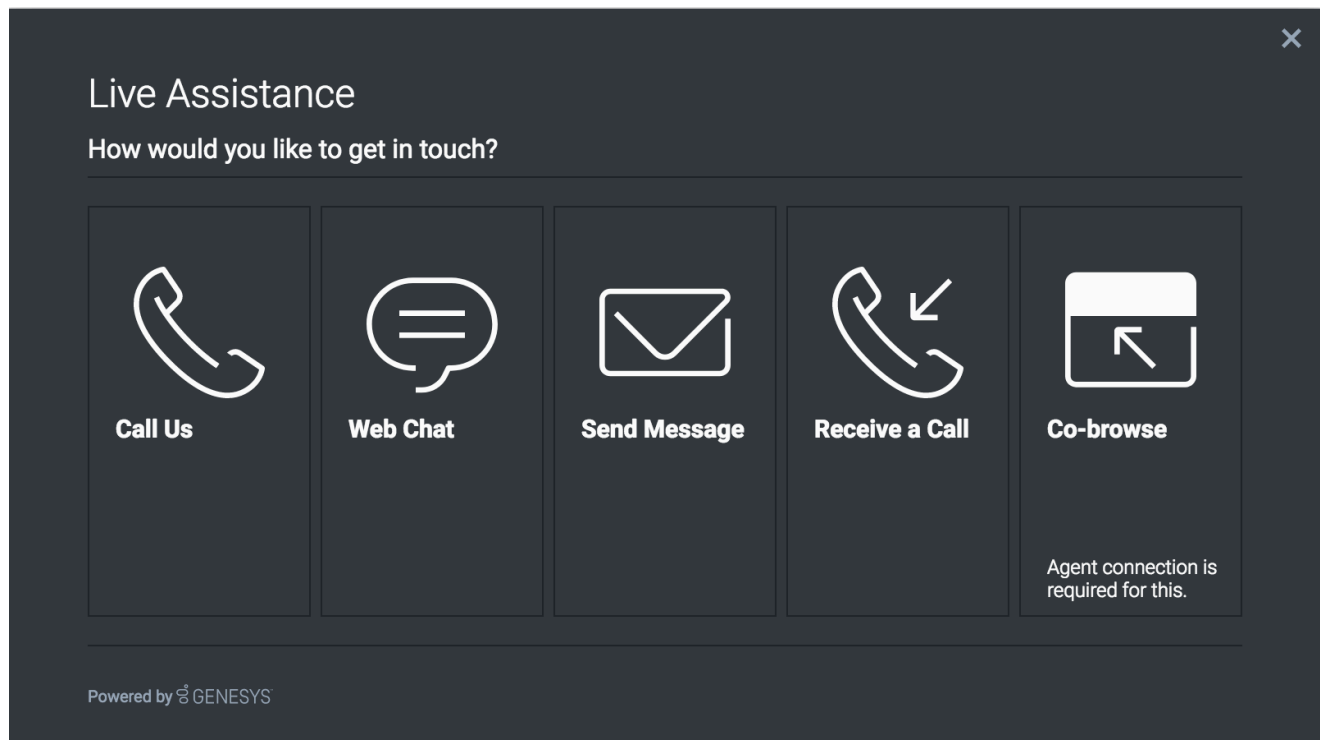
Important

The global bus object is a debug 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('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



- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

The ChannelSelector widget provides a configurable list of channels as an entry point for customers to contact customer service. In addition to showing multiple channels, ChannelSelector can show the Estimated Wait Time (EWT) for each channel when configured. You can also configure channels to hide or show as disabled based on EWT value. Channels are not limited to Genesys Widgets, you can add your own custom channels to launch applications or open new windows as necessary.

See the screenshots below and visit the [configuration](#) page for more information.

Usage

ChannelSelector can be launched manually by the following methods:

- Calling the **command** "ChannelSelector.open"
- 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. Please see the **configuration** for examples and information on how to setup your own custom channels.

Customization

All static text shown in the ChannelSelector Widget is fully customizable and **localizable** by adding entries into your **configuration** and **localization** options.

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

Namespace

Channel Selector plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	channelselector
i18n - Localization	channelselector
CXBus - API Commands & API Events	ChannelSelector
CSS	.cx-channel-selector

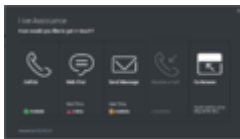
Mobile Support

ChannelSelector supports both desktop and mobile devices. Like all Genesys Widgets, there are two main modes: Desktop & 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.

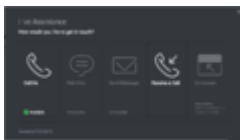
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

"Dark" Theme



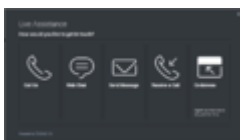
Desktop overlay view showing
EWT available



Desktop overlay view showing
EWT maximum & unavailable



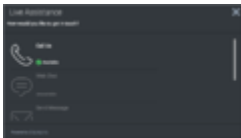
Desktop overlay view showing
EWT minimums



Desktop overlay view showing
Co-browse channel}



Mobile full-screen view showing
EWT available



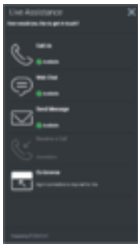
Mobile full-screen view showing
EWT maximum & unavailable



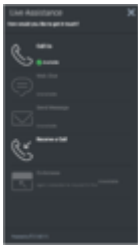
Mobile full-screen view showing
EWT maximum



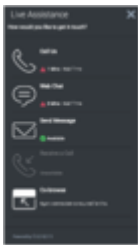
Mobile full-screen view showing
EWT minimum



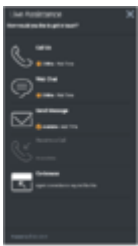
Mobile full-screen view showing
EWT available



Mobile full-screen view showing
EWT maximum & unavailable



Mobile full-screen view showing
EWT maximum



Mobile full-screen view showing
EWT minimum

"Light" Theme



Desktop overlay view showing
EWT available



Desktop overlay view showing
EWT maximum & unavailable



Desktop overlay view showing
EWT minimums



Desktop overlay view showing
Co-browse channel



Mobile full-screen view showing
EWT available



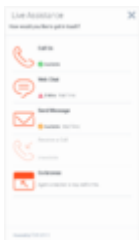
Mobile full-screen view showing
EWT maximum & unavailable



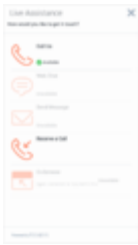
Mobile full-screen view showing
EWT maximum



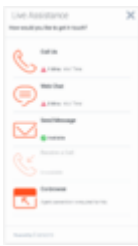
Mobile full-screen view showing
EWT minimum



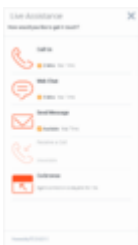
Mobile full-screen view showing
EWT available



Mobile full-screen view showing
EWT maximum & unavailable



Mobile full-screen view showing
EWT maximum



Mobile full-screen view showing
EWT minimum

Configuration

Description

ChannelSelector shares the configuration namespace '_genesys.widgets.channelselector'. ChannelSelector has UI options to enable/disable channels, hide channels, add new channels and display Estimated Waiting Time details. All the channels are displayed based on the array of objects order defined in channels config. To hide a particular channel, simply remove corresponding array object.

Example

```
window._genesys.widgets.channelselector = {
  ewtRefreshInterval: 10,
  channels: [{
    enable: true,
    clickCommand: 'CallUs.open',
    readyEvent: 'CallUs.ready',
    displayName: 'Call Us',
    i18n: 'CallusTitle',
    icon: 'call-outgoing',
    html: '<img src='http://placeholder.it/100x100'>',
    ewt: {
      display: true,
      queue: 'callus_ewt_test_eservices',
      availabilityThresholdMin: 300,
      availabilityThresholdMax: 480,
      hideChannelWhenThresholdMax: false
    }
  },
  {
    enable: true,
    clickCommand: 'WebChat.open',
    readyEvent: 'WebChat.ready',
    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: 'SendMessage.open',
      readyEvent: 'SendMessage.ready',
      displayName: 'Send Message',
      i18n: 'EmailTitle',
      icon: 'email',
      html: '',
      ewt: {
        display: true,
        queue: 'email_ewt_test_eservices',
        availabilityThresholdMin: 300,
        availabilityThresholdMax: 480,
        hideChannelWhenThresholdMax: false
      }
    },

    {
      enable: true,
      clickCommand: 'Callback.open',
      readyEvent: 'Callback.ready',
      displayName: 'Receive a Call',
      i18n: 'CallbackTitle',
      icon: 'call-incoming',
      html: '',
      ewt: {
        display: true,
        queue: 'callback_ewt_test_eservices',
        availabilityThresholdMin: 300,
        availabilityThresholdMax: 480,
        hideChannelWhenThresholdMax: false
      }
    },

    {
      enable: true,
      name: "CoBrowse",
      clickCommand: "CoBrowse.open",
      readyEvent: "CoBrowse.ready",
      displayName: "Co-browse",
      i18n: "CobrowseTitle",
      icon: "cobrowse",
      html: ''
    }
  ]
};

```

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	none	Always

Name	Type	Description	Default	Required
		for opening a particular widget when clicked on this channel.		
channels[].readyEvent	string	Subscribes to this ready event published by a plugin and enables the channel when that plugin is ready.	none	n/a
channels[].displayName	string	A channel name to display on ChannelSelector widget.	none	Always
channels[].i18n	string	To support localization of channel display name, this takes a key parameter of channelselector section in language pack file. Overrides above displayName.	none	n/a
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
chat.ewtMax	number	If EWT is greater than ewtMin and less this value (seconds), Wait time is shown with red alert icon.	480	n/a
channels[].ewt.display	boolean	To display estimated waiting time details.	true	n/a
channels[].ewt.queue	string	EWT service channel virtual queue.	none	Always
channels[].ewt.availabilityThresholdMin	number	If EWT is greater than 0 and less this minimum	300	n/a

Name	Type	Description	Default	Required
		threshold value (seconds), estimated waiting time is shown with yellow warning icon.		
channels[].ewt.availabilityThresholdMax	int	If EWT is greater than minimum threshold value and less maximum threshold value (seconds), estimated waiting time is shown with red alert icon.	480	n/a
channels[].ewt.hideChannelWhenThresholdMax	boolean	Hides this channel when estimated waiting time is greater than maximum threshold value.	true	n/a

Localization

Important

For information on how to setup localization, please refer to the [Localization Guide](#)

Usage

'channelselector' namespace should be used when defining localization strings for ChannelSelector plugin in your i18n JSON file.

In the example below, we demonstrate defining new strings for the 'en' (English) language. You may 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": {
    "channelselector": {
      "Title": "Live Assistance",
      "SubTitle": "How would you like to get in touch?",
      "WaitTimeTitle": "Wait Time",
      "AvailableTitle": "Available",
      "UnavailableTitle": "Unavailable",
      "CobrowseButtonText": "CobrowseSubTitle",
      "CallbackTitle": "Receive a Call",
      "CobrowseSubTitle": "Agent connection is required for this."
    }
  }
}
```

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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	When ChannelSelector is successfully closed	n/a
rejected	When 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	When ChannelSelector widget is successfully opened	n/a
rejected	When ChannelSelector widget is already open	'Already open'

configure

Modify the configuration for ChannelSelector.

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) defined.
channels	array	Array containing each channel configuration object. The order of channels are 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 clicked on this channel.
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 ChannelSelector widget.
channels[].i18n	string	To support localization of channel display name, this takes a key parameter of channelselector section in 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 estimated waiting time details.
channels[].ewt.queue	string	EWT service channel virtual queue name.
channels[].ewt.availabilityThreshold	number	If EWT is greater than 0 and less than this minimum threshold value

Option	Type	Description
		(seconds), estimated waiting time is shown with yellow warning icon.
channels[].ewt.availabilityThreshold	Number	If EWT is greater than minimum threshold value and less maximum threshold value (seconds), estimated waiting time is shown with red alert icon.
channels[].ewt.hideChannelWhenThreshold	Boolean	Hides this channel when estimated waiting time is greater than maximum threshold value.

Resolutions

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

displayStats

Estimated Waiting Time and availability details are displayed 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	When Estimated Waiting Time is displayed successfully.	n/a
rejected	When StatsService fails to	'Unable to display EWT Stats in

Status	When	Returns
	retrieve EWT data.	ChannelSelector'
rejected	When enableEwt config is disabled or when required channel plugins are not ready.	'Either EWT config is disabled or plugins not yet ready'

disableStats

UI is cleared of any Estimated Waiting time. Fetching it for the defined time interval is also disabled.

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

enableStats

UI is shown back with estimated Waiting time and availability details. Fetching it for the defined time interval is also enabled.

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	When ChannelSelector widget is successfully opened.	n/a
rejected	When Estimated Waiting time details are already displayed.	'Stats already enabled'
rejected	When ChannelSelector widget is not opened.	'ChannelSelector not opened to enable stats details'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.



Important

The global bus object is a debug 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('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

ChatDeflection

 Live Chat 



John Doe
What is Genesys Knowledge Center?
11:21 AM






Knowledge Center
While waiting for an Agent to connect, here are the most relevant answers to your query:

- [What Is Genesys Knowledge Center?](#)
- [What components are included in Genesys Knowledge Center?](#)

Powered by  GENESYS

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

Important

ChatDeflection Widget is available starting from the 8.5.004.09 version of the Genesys Widgets

The ChatDeflection widget allows a customer to address a question while waiting for a customer service agent to join a live chat. ChatDeflection does not introduce new UI, it is just adding additional functionality to the WebChat widget. ChatDeflection widget uses the KnowledgeCenterService widget to match a customer's question to the corporate knowledge base and come up with the most relevant knowledge for that question. ChatDeflection stops any interactions with the customer as soon as the customer service agent joins the live chat session. The customer service agent who joins the session after the deflection attempt, now has some context of the customer issue ready for review, as well as the information on the suggested knowledge and the customer's interactions with it.

Usage

ChatDeflection will be launched automatically when the live chat session started. It can also be manually enabled or disabled by the following methods:

- Enabled by calling the **command** "ChatDeflection.enable"
- Disabled by calling the **command** "ChatDeflection.disable"

Deployment Notes

ChatDeflection Configuration

ChatDeflection utilizes the Genesys Knowledge Center Server Knowledge API accessible through the KnowledgeCenterService widget.

Does deflection attempt will be shown in the transcript?

The ChatDeflection widget has several different modes of reporting chat deflection actions to the chat transcript:

- none - deflection actions are not visible in transcript
- readable (default) - deflection actions shown in human-readable format in the chat transcript
- JSON - deflection actions stored as the JSON object with all the technical details

Customization

All static text shown during chat deflection session is fully customizable and **localizable** by adding entries into your **configuration** and **localization** options.

ChatDeflection supports Themes. You may create and register your own themes for Genesys Widgets.

Namespace

Chat Deflection plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	knowledgecenter
i18n - Localization	knowledgecenter
CXBus - API Commands & API Events	ChatDeflection
CSS	.cx-kc-article

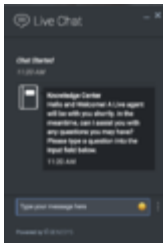
Mobile Support

ChatDeflection 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, ChatDeflection switches to special fullscreen templates that are optimized for both portrait and landscape orientations.

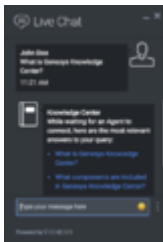
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

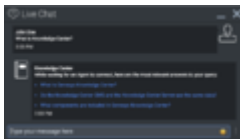
"Dark" Theme



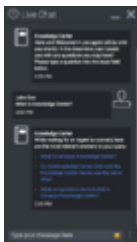
Mobile fullscreen view in portrait orientation showing deflection invitation message



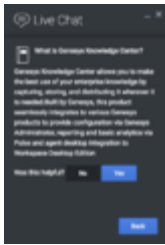
Desktop docked view showing deflection response



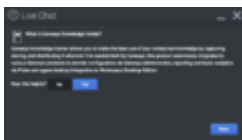
Mobile fullscreen view in Landscape orientation showing deflection response



Mobile fullscreen view in portrait orientation showing deflection response



Desktop docked view showing document details (since 8.5.004.19)

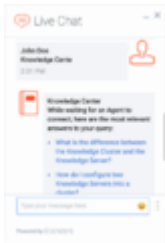


Mobile fullscreen view in portrait orientation showing document details (since 8.5.004.19)

"Light" Theme



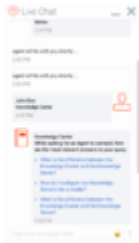
Mobile fullscreen view in portrait orientation showing deflection invitation message



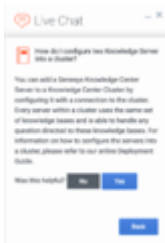
Desktop docked view showing deflection response



Mobile fullscreen view in
Landscape orientation showing
deflection response



Mobile fullscreen view in portrait
orientation showing deflection
response



Desktop docked view showing
document details (since
8.5.004.19)



Mobile fullscreen view in portrait
orientation showing document
details (since 8.5.004.19)

Configuration

Description

cx-chat-deflection uses '_genesys.cxwidget.knowledgecenter' configuration namespace and has connection and chat-deflection options.

Example

```
window._genesys.cxwidget.knowledgecenter = {  
  deflection:{  
    enabled:true,  
    agentTranscript:'readable'  
  }  
}
```

Options

Name	Type	Description	Default	Required
enabled	boolean	Enables/disables chat deflection functionality. Can be changed programmatically using enable/disable commands of the widget.	true	
agentTranscript	string	Defines how the Knowledge Center responses will be stored in chat transcript. Valid values: none, readable (default)	readable	
workspace.enabled	boolean	Enables/disables context data of chat deflection to be attached to the chat interaction user data. This data is consumed by Workspace to show agent information about	true	

Name	Type	Description	Default	Required
		knowledge session Note: available since 8.5.004.19		
workspace.sessionKeystring		User data key that will contain knowledge session id associated with the deflection session. Valid values: valid user data key name Note: available since 8.5.004.19	gks_session	
workspace.languageKeystring		User data key that will contain language id associated with the deflection session. Valid values: valid user data key name Note: available since 8.5.004.19	gks_lang	
workspace.questionKeystring		User data key that will contain last searched question. Valid values: valid user data key name Note: available since 8.5.004.19	gks_question	
reporting.enabled	boolean	Enables/disables chat deflection progress status to be attached to the chat interaction user data. This data can be used in reporting to analyze deflection sessions and their outcomes Note: available since 8.5.004.19	true	

Localization

Important

For information on how to setup localization, please refer to the [Localization Guide](#)

Usage

'knowledgecenter' namespace should be used when defining localization strings for ChatDeflection plugin in your i18n JSON file.

In the example below, we demonstrate defining new strings for the 'en' (English) language. You may 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": {
    "knowledgecenter": {
      "KnowledgeAgentName": "Knowledge Center",
      "WelcomeMessage": "Hello and Welcome! A Live agent will be with you
shortly. In the meantime, can I assist you with any questions you may have? Please type a
question into the input field below.",
      "SearchResult": "While waiting for an Agent to connect, here are the
most relevant answers to your query:",
      "NoDocumentsFound": "I'm sorry. No articles matched your question.
Would you like to ask another question?",
      "Yes": "Yes",
      "No": "No",
      "Back": "Back",
      "FeedbackQuestion": "Was this helpful?",
      "FeedbackAccept": "Yes",
      "FeedbackDecline": "No",
      "ArticleHelpfulnessYes": "Article Helpfulness - 'Yes'",
      "ArticleHelpfulnessYesDesc": "Great! We're very pleased to hear that
the article assisted you in your search. Have a great day!",
      "ArticleHelpfulnessNo": "Article Helpfulness - 'No'",
      "ArticleHelpfulnessNoDesc": "We're sorry that the article wasn't a
good match for your search. We thank you for your feedback!",
      "TranscriptMarker": "KnowledgeCenter: ",
      "SearchMessage": "Search with query '<%SearchQuery%>'↓",
      "VisitMessage": "Visit for document '<%VisitQuery%>'",
      "AnsweredMessage": "Results for query '<%AnsweredQuery%>' have been
marked as relevant.",
    }
  }
}
```

```
        "UnansweredMessage": "Results for query '<%UnansweredQuery%>' have  
been marked as unanswered.",  
        "PositiveVoteMessage": "Positive voting for document '<%VoteQuery%>'",  
        "NegativeVoteMessage": "Negative voting for document '<%VoteQuery%>'"  
    }  
}
```

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('ChatDeflection.enable');
```

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('ChatDeflection.configure', {  
    enable: true,  
    agentTranscript: 'readable'  
}).done(function(e){  
    // ChatDeflection configured successfully  
}).fail(function(e){  
    // ChatDeflection failed to configure  
});
```

Options

Option	Type	Description
enable	boolean	Enables/disables chat deflection functionality. Can be changed programmatically using enable/disable commands of the widget.
agentTranscript	string	Defines how the Knowledge Center responses will be stored in chat transcript.

Resolutions

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

enable

Enable chat deflection

Example

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

Resolutions

Status	When	Returns
resolved	Chat deflection has been enabled	n/a

disable

Disable chat deflection

Example

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

Resolutions

Status	When	Returns
resolved	Chat deflection has been disabled	n/a

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('ChatDeflection.ready', function(e){});
```

Name	Description	Data
ready	ChatDeflection is initialized and ready to accept commands	n/a
enabled	ChatDeflection has been enabled. It will happen for ongoing and any new chat session	n/a
disabled	ChatDeflection has been disabled. It is stopped for any ongoing session as well as for future chat sessions	n/a
started	ChatDeflection attempt has been started for current active session	n/a
ended	ChatDeflection attempt has been ended for the current session	n/a

Common

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',
  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'

```
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. For more details, refer: http://iscrolljs.com/#configuring

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 false. 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 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 <a href='www.genesys.com' target='_self'>www.genesys.com</a>
```

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 'debug' or 'error' log type, use 'log' instead.

Common.sanitizeHTML(string)

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

Example

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

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

```
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 = $('<div><button class='i18n' data-message='CustomButton001'>
</button></div>');
```

```
Common.updateTemplateI18n(ndContainer, {CustomButton001: 'Accept'});
```

```
// ndContainer == <div><button class='i18n' data-
message='CustomButton001'>Accept</button></div>
```

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 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_continer'));
```

Arguments

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

Argument	Type	Description
		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')
```

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 like to display. Note: Refer to Common.debugIcons method to find out all the icons names that widgets supports.

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'))
```


Arguments

Argument	Type	Description
element	jQuery DOM Selector	An html container where waiting symbol should appear. This adds a class name 'cx-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 till the 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.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

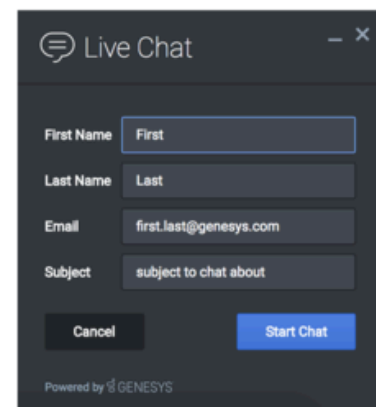
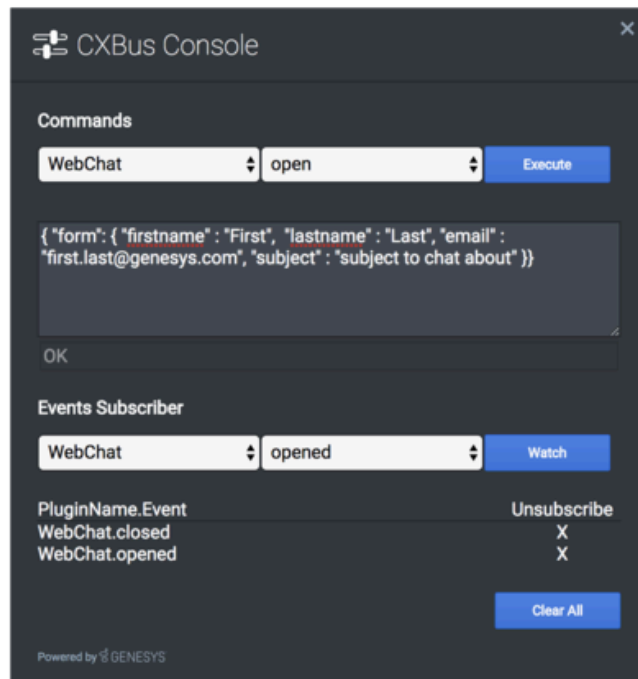
'convert date timestamp to return time in 12 hrs format'

```
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.

Console



- [Configuration](#)
- [Localization](#)
- [Commands](#)
- [Events](#)

Overview

The Console Widget is a tool for debugging commands and events on the widget bus. You can test, debug, or demo all commands using dynamically populated lists and create event watchlists that alert you when an event has fired.

Console provides an easy to use interface for debugging the widget bus that compliments 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, as you left it. It is a great tool for getting to know the widget bus, the API for each widget, and debugging issues.

Usage

WebChat can be launched manually by the following methods:

- Calling the command "Console.open"
- Configuring settings to show Console upon opening the browser.
- Creating 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 setup localization, please refer to the [Localization Guide](#)

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 own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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	When Console is successfully opened	n/a
rejected	When 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	When Console successfully closed	n/a
rejected	When Console is already closed	'Already closed'

configure

Modify configuration options for Console. See configuration page for Console

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	When Console configuration is provided	n/a
rejected	When no configuration provided	'Invalid Configuration'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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	

KnowledgeCenterService

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

Important

KnowledgeCenterService Widget is available starting from the 8.5.004.09 version of the Genesys Widgets

KnowledgeCenterService exposes a high-level API for utilizing Genesys Knowledge Center services. You can use these services for exposing corporate knowledge on the web site via standard widgets or for developing your own custom knowledge-aware widgets. KnowledgeCenterService provides a unified way for all widgets utilizing bus communication to access the corporate knowledge easily.

Usage

KnowledgeCenterService and the matching [Search](#) and [ChatDeflection](#) widgets work together right out of the box and they share the same configuration object. Using Search or ChatDeflection requires use of KnowledgeCenterService.

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

Namespace

Knowledge Center Service plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	knowledgecenter
CXBus - API Commands & API Events	KnowledgeCenterService

Customization

KnowledgeCenterService has many configuration options but no customization options. It is meant as a plug-n-play type of plugin and works as-is.

Configuration

Description

KnowledgeCenterService, Search and ChatDeflection share the configuration namespace '_genesys.cxwidget.knowledgecenter'. KnowledgeCenterService defines connection options and default values for content retrieval options while other plugins have configuration specific for every particular function.

Example

```
window._genesys.cxwidget.knowledgecenter = {  
  host: 'http://gks-dep-stbl:9092/gks-server/v1',  
  knowledgebases: ['knowledgefaq', 'knowledgearticles'],  
  lang: 'en',  
  media: 'chat',  
  maxTrendingResults: 5,  
  maxSearchResults: 3,  
  apiClientId: 'widget',  
  apiClientMediaType: 'selfservice'  
}
```

Options

Name	Type	Description	Default	Required
host	string	Knowledge Center Server API URL	n/a	Always
knowledgebases	object	List of knowledge base IDs that knowledge will be searched in. Empty value will allow search in all knowledgebases that are publicly available		Always
lang	string	Language in which knowledge search will be executed	en	Always
media	string	Media that content needs to be searched for. Empty value allows any	all	

Name	Type	Description	Default	Required
		available content to be searched.		
maxTrendingResults	number	Maximum number of documents in trending response	5	
maxSearchResults	number	Maximum number of documents in search response	3	
apiClientId	string	Client ID of the application using knowledge (for reporting purposes).	cxwidget	
apiClientMediaType	string	Media type that knowledge uses (for reporting purposes).	selfservice	
tenantId	number	Specifies tenantId that needs to be used in requests to Knowledge Center. If not defined, this parameter is not added to requests.	not defined	

Localization

No Localization options

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('KnowledgeCenterService.search');
```

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('KnowledgeCenterService.configure', {  
    host: 'http://localhost:8080/foo/bar',  
    knowledgebases: [ 1, 2, 3, 4, 5 ],  
    lang: 'eng'  
}).done(function(e){  
    // KnowledgeCenterService configured successfully  
}).fail(function(e){  
    // KnowledgeCenterService failed to configure  
});
```

Options

Option	Type	Description
host	string	Knowledge Center Server API URL
knowledgebases	object	Array of knowledge base IDs for all further requests
lang	string	Default language for all further requests
media	string	Media that content needs to be searched for.
apiClientId	string	Default Client ID of application using knowledge
apiClientMediaType	string	Default Media knowledge is used on

Resolutions

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

getTrending

Fetch trending documents

Example

```
oMyPlugin.command('KnowledgeCenterService.getTrending', {size: 25}).done(function(e){  
    // KnowledgeCenterService got trending documents successfully  
    // e == Object with trending categories and documents  
}).fail(function(e){  
    // KnowledgeCenterService failed to get trending documents  
});
```

Options

Option	Type	Description
size	number	Maximum number of returned items

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	Object with trneding categories
rejected	When KC Server returns error	'KC Server error'

search

Search documents relevant to query

Example

```
oMyPlugin.command('KnowledgeCenterService.search', {
    query: 'topic',
    size: 10,
    categories: [ 1, 2, 3, 4, 5 ]
}).done(function(e){
    // KnowledgeCenterService search executed successfully
    // e == Object with search results
}).fail(function(e){
    // KnowledgeCenterService failed to execute search
});
```

Options

Option	Type	Description
query	string	Search query
size	number	Maximum number of returned items
categories	object	Array of Category IDs for

Option	Type	Description
		additional filter
knowledgebases	object	Array of knowledge base IDs for all further requests. Overwrites knowledgeCenterServer widget settings
lang	string	Default language for all further requests. Overwrites knowledgeCenterServer widget settings
media	string	Media that content needs to be searched for. Overwrites knowledgeCenterServer widget settings

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	n/a
rejected	When KC Server returns error	'KC Server error'

getSuggestions

Search suggestions for autocomplete functionality

Example

```
oMyPlugin.command('KnowledgeCenterService.getSuggestions', {
    query: 'topic',
    size: 10,
    categories: [ 1, 2, 3, 4, 5 ]
}).done(function(e){
    // KnowledgeCenterService got suggested documents successfully
    // e == Object with suggestions
}).fail(function(e){
    // KnowledgeCenterService failed to get suggested documents
});
```

Options

Option	Type	Description
query	string	Search query
size	number	Maximum number of returned items
categories	object	Array of Categories ID for additional filter

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	n/a
rejected	When KC Server returns error	'KC Server error'

getCategories

Get list of categories

Example

```
oMyPlugin.command('KnowledgeCenterService.getCategories').done(function(e){  
    // KnowledgeCenterService got categories successfully  
    // e == Object with categories  
}).fail(function(e){  
    // KnowledgeCenterService failed to get categories  
});
```

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	Object containing categories
rejected	When KC Server returns error	'KC Server error'

getFullContent

Get full document content

Example

```
oMyPlugin.command('KnowledgeCenterService.getFullContent', {
    docId: '12345',
    kbId: '1'
}).done(function(e){
    // KnowledgeCenterService got full content successfully
    // e == Object with content of a document
}).fail(function(e){
    // KnowledgeCenterService failed to get full content
});
```

Options

Option	Type	Description
docId	string	Document ID
kbId	string	Knowledge base ID where the document is located
lang	string	Default language for all further requests

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	n/a
rejected	When KC Server returns error	'KC Server error'

visit

Mark a document as opened

Example

```
oMyPlugin.command('KnowledgeCenterService.visit', {
    docId: '12345',
    kbId: '1'
}).done(function(e){
    // KnowledgeCenterService marked as visited successfully
}).fail(function(e){
    // KnowledgeCenterService failed to mark as visited
});
```

Options

Option	Type	Description
docId	string	Document ID
kbId	string	Knowledge base ID where the document is located
query	string	Used query for prior search
categories	object	Used categories filter for prior search

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	n/a
rejected	When KC Server returns error	'KC Server error'

vote

Provide relevancy feedback (relevant/irrelevant)

Example

```
oMyPlugin.command('KnowledgeCenterService.vote', {
    docId: '12345',
    kbId: '1',
```

```
        query: 'search',
        relevant: 'true'
    }).done(function(e){
        // KnowledgeCenterService voted successfully
    }).fail(function(e){
        // KnowledgeCenterService failed to vote
    });
```

Options

Option	Type	Description
docId	string	Document ID
kbId	string	Knowledge base ID where the document is located
query	string	Used query for prior search
categories	object	Used categories filter for prior search
relevant	boolean	Whether the document was relevant

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	n/a
rejected	When KC Server returns error	'KC Server error'

rating

Add quality rating and comment to a document

Example

```
oMyPlugin.command('KnowledgeCenterService.rating', {
    docId: '12345',
    kbId: '1',
    comment: 'text',
    rating: 5
});
```

```
}).done(function(e){
    // KnowledgeCenterService rated successfully
}).fail(function(e){
    // KnowledgeCenterService failed rate
});
```

Options

Option	Type	Description
docId	string	Particular document ID
kbId	string	Knowledge base ID where the document is located
comment	string	Comment
rating	number	Rating: 1, 2, 3, 4 or 5 stars

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	n/a
rejected	When KC Server returns error	'KC Server error'

addRating

Submit quality rating for previously submitted relevancy feedback

Example

```
oMyPlugin.command('KnowledgeCenterService.addRating', {
    voteId: '123',
    comment: 'text',
    rating: 5
}).done(function(e){
    // KnowledgeCenterService added rating successfully
}).fail(function(e){
```

```
}); // KnowledgeCenterService failed add rating
```

Options

Option	Type	Description
voteId	string	Vote ID returned after 'vote' command
comment	string	Comment
rating	number	Rating: 1, 2, 3, 4 or 5 stars

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	n/a
rejected	When KC Server returns error	'KC Server error'

unanswered

Mark search result as the one that does not contain relevant documents

Example

```
oMyPlugin.command('KnowledgeCenterService.unanswered', {
    kbId: '12345',
    query: 'text'
}).done(function(e){
    // KnowledgeCenterService marked search result as irrelevant successfully
}).fail(function(e){
    // KnowledgeCenterService failed mark search result as irrelevant
});
```

Options

Option	Type	Description
kbld	string	Knowledge base ID where search were executed
query	string	Used query
categories	object	Used categories filter

Resolutions

Status	When	Returns
resolved	When KC Server returns appropriate response	n/a
rejected	When KC Server returns error	'KC Server error'

sessionInfo

Retrieves parameters of the current knowledge session

Example

```
oMyPlugin.command('KnowledgeCenterService.sessionInfo').done(function(e){  
    // KnowledgeCenterService got session info successfully  
    // e == {sessionId: "<sessionId>", language: "<languageCode>", media: "<mediaType>"}  
}).fail(function(e){  
    // KnowledgeCenterService failed to get session info  
});
```

Resolutions

Status	When	Returns
resolved	When there is existing knowledge session	{sessionId: "<sessionId>", language: "<languageCode>", media: "<mediaType>"}
rejected	When knowledge session is not established yet	'Knowledge center session is not started yet'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('KnowledgeCenterService.ready', function(e){ /* sample code */ });
```

Name	Description	Data
ready	The KnowledgeCenterService widget is loaded.	n/a
online	The KnowledgeCenterService widget is configured and ready to execute requests.	n/a
sessionChanged	The session started or one of it's parameters had changed.	{sessionId: "<sessionId>", language: "<languageCode>", media: "<mediaType>"}

Overlay

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

The Overlay plugin provides an Overlay window control that widgets can inject their UI into. The Overlay plugin accepts the HTML UI and puts it inside an Overlay control and displays the UI onscreen in a uniform overlay window fashion. This prevents individual widgets from managing the overlay themselves and each widget's UI can be move between different container types.

Overlay provides these benefits

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

Usage

Overlay is very 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`.

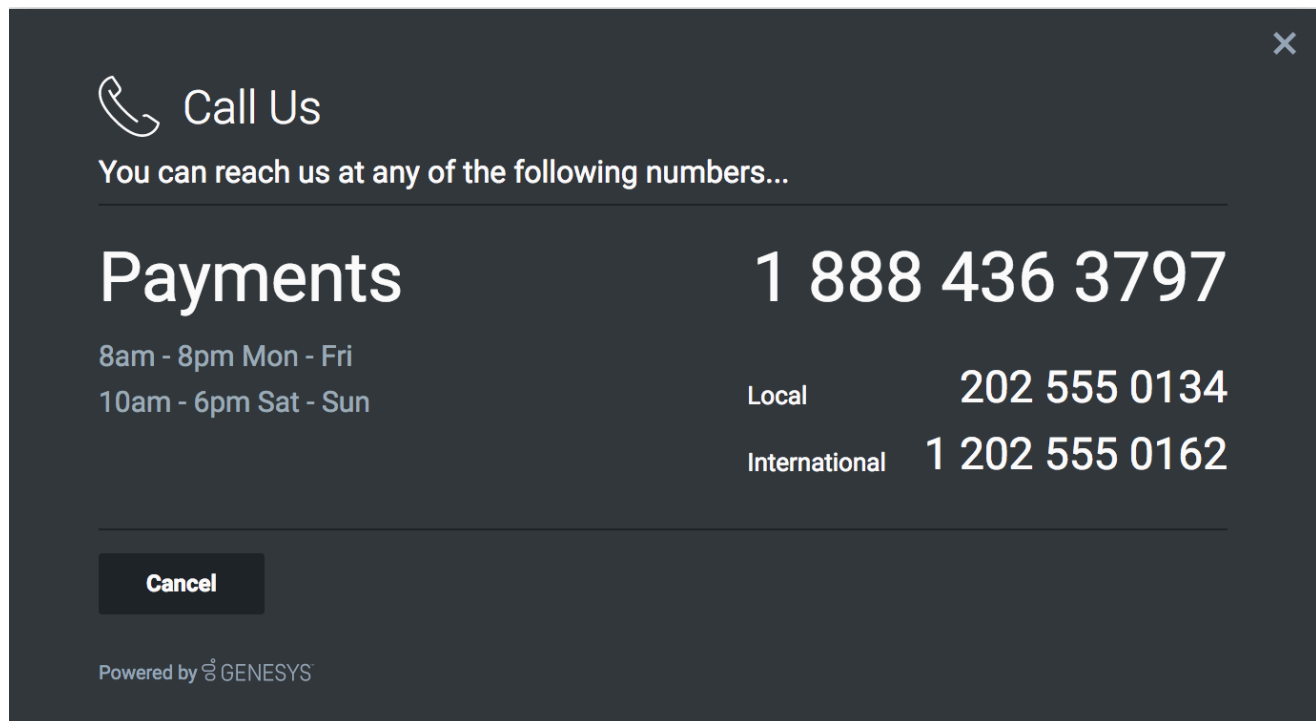
Customization

Overlay does not have customization options.

Mobile Support

Overlay will automatically apply 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.

Screenshots



Configuration

No configuration options

Localization

No localization options

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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: '<div>Template</div>',  
    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	When set to true, overlay cannot be closed by other plugins.

Resolutions

Status	When	Returns
resolved	When overlay is successfully opened	{html: <template>, events: <Object>, group: <String> }
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.'

close

Closes the Overlay UI. Publishes the appropriate custom close event for current overlay being closed.

Example

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

Resolutions

Status	When	Returns
resolved	When Overlay is successfully closed.	n/a

Status	When	Returns
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 own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

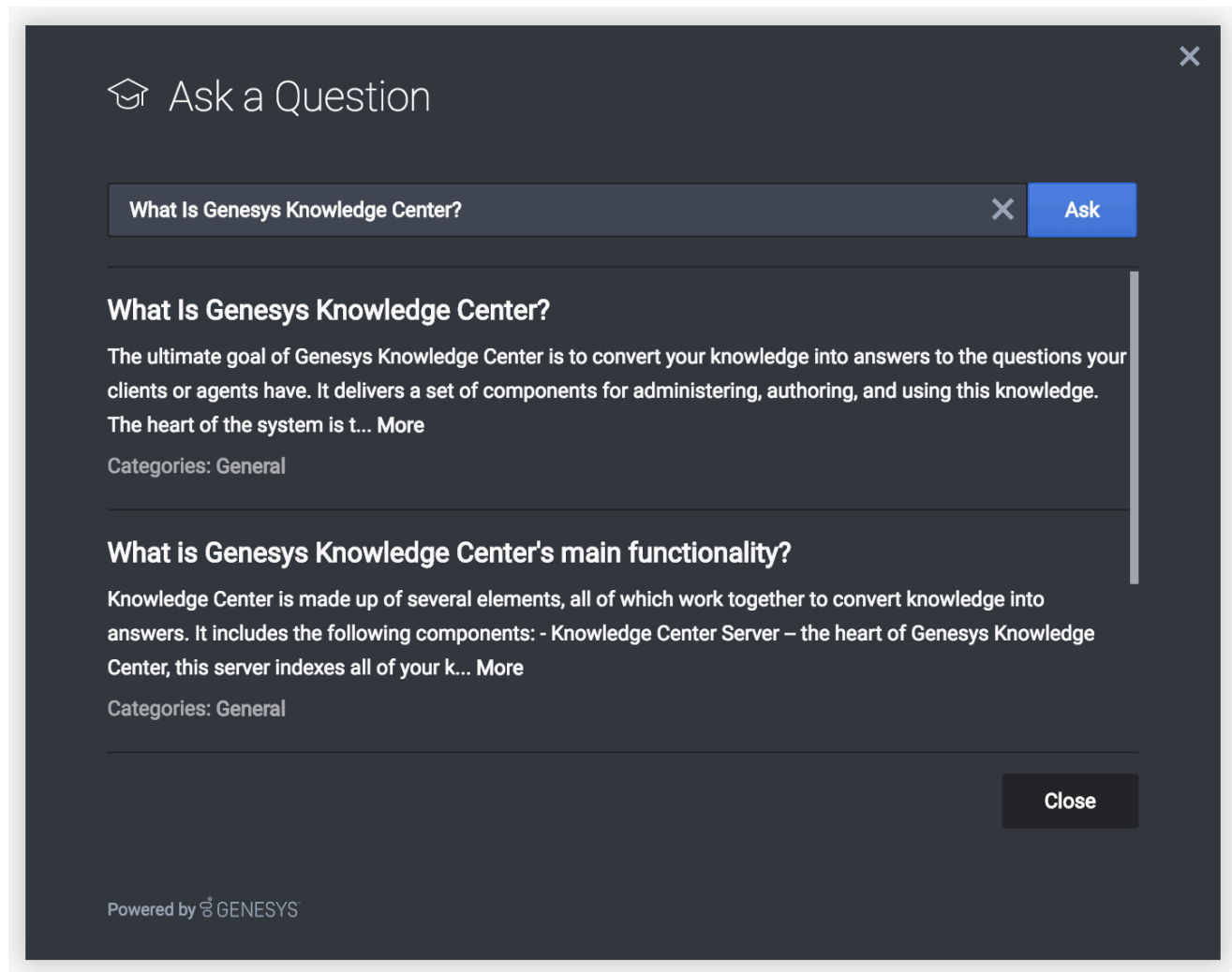
Important

The global bus object is a debug 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('Overlay.ready', function(e){});
```

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

Search



- [Commands](#)
- [Events](#)
- [API Configuration](#)
- [API Localization](#)

Overview

Important

Search Widget is available starting from the 8.5.004.09 version of the Genesys Widgets

The Search widget allows a customer to address his question to the corporate knowledge. The UI appears within the page. Customers can ask a question (search), review provided results, and provide feedback on whether the results addressed the problem.

Usage

Search can be launched manually by the following methods:

- Calling the **command** "Search.open"
- Enable the built-in launcher button for Search that appears on the right side of the screen
- Create your own custom button or link to open Search (using the "Search.open" command)

Namespace

Search plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	knowledgecenter.search
i18n - Localization	knowledgecenter
CXBus - API Commands & API Events	Search
CSS	.cx-search

Deployment Notes

Search Configuration

Genesys Search utilizes the Genesys Knowledge Center Server Knowledge API.

For more information on Genesys Knowledge Center and its APIs, please see the following links:

- [Genesys Knowledge Center documentation](#)
- [Knowledge API](#)
- [Genesys Knowledge Center Developer's Guide](#)

Can I open the Search Widget with search results pre-populated?

The Search Widget allows "Search.open" command to execute with optional parameter "question" which contains the initial question the Search Widget needs in order to pre-populate answers.

Customization

All static text shown in the Search Widget are fully customizable and **localizable** by adding entries into your **configuration** and **localization** options.

Search supports Themes. You may create and register your own themes for Genesys Widgets.

Mobile Support

Search 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, Search switches to special fullscreen 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

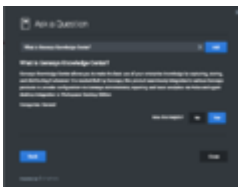
"Dark" Theme



Desktop Search Widget with contextual help when typing



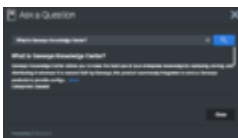
Desktop Search Widget showing search results



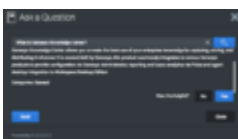
Desktop Search Widget showing document details



Mobile fullscreen view in portrait orientation showing search results



Mobile fullscreen view in landscape orientation showing search results



Mobile fullscreen view in
landscape orientation showing
document details

"Light" Theme



Desktop Search Widget with
contextual help when typing



Desktop Search Widget showing
search results



Desktop Search Widget showing
document details



Mobile fullscreen view in portrait
orientation showing search
results



Mobile fullscreen view in landscape orientation showing search results



Mobile fullscreen view in landscape orientation showing document details

Configuration

Description

Search reads its configuration from the subnode of the KnowledgeCenterService configuration namespace '_genesys.widgets.knowledgecenter.search'.

Example

```
window._genesys.widgets.knowledgecenter.search = {SearchButton: {
    enabled: true,
    template: <div class='cx-icon' data-icon='search'></div>,
    effect: 'fade',
    openDelay: 1000,
    effectDuration: 300
  }
};
```

Options

Name	Type	Description	Default	Required
SearchButton.enabled	boolean	Enable/disable search button on screen.	false	
SearchButton.template	string	Custom HTML string template for search button.	<div class='cx-widget cx-search-button cx-side-button' data-message='SearchButton' data-gcb-service-node='true'>	
SearchButton.effect	string	Type of animation effect when revealing chat	fade	

Name	Type	Description	Default	Required
		button. 'slide' or 'fade'.		
SearchButton.openDelay	Number	Number of milliseconds before displaying chat button on screen.	1000	
SearchButton.effectDuration	Number	Length of animation effect in milliseconds	300	

Localization

Important

For information on how to setup localization, please refer to the [Localization Guide](#)

Usage

'knowledgecenter' namespace should be re-used when defining localization strings for Search plugin in your i18n JSON file.

In the example below, we demonstrate defining new strings for the 'en' (English) language. You may 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": {
    "knowledgecenter": {
      "SidebarButton": "Search",
      "SearchButton": "Search",
      "Title": "Ask a Question",
      "Ask": "Ask",
      "Close": "Close",
      "Categories": "Categories",
      "NoResults": "No Results",
      "NoResultsTextUnder": "We're sorry but we could not find a suitable
answer for you.",
      "NoResultsTextRephrase": "Could you please try rephrasing the
question?",
      "WasThisHelpful": "Was this helpful?",
      "Yes": "Yes",
      "No": "No",
      "ArticleHelpfulnessYes": "Article Helpfulness - 'Yes'",
      "ArticleHelpfulnessYesDesc": "Great! We're very pleased to hear that
the article assisted you in your search. Have a great day!",
      "ArticleHelpfulnessNo": "Article Helpfulness - 'No'",
      "ArticleHelpfulnessNoDesc": "We're sorry that the article wasn't a
good match for your search. We thank you for your feedback!",
      "TypeYourQuestion": "Type your question",
      "Back": "Back",
      "More": "More",
      "Error": "Error!",
      "GKCIUnavailable": "Knowledge Center Server is currently not
```

```
available"
    }
}
```


API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('Search.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.

Example

```
oMyPlugin.command('Search.configure', {  
    enabled: false,  
    hideDuringInvite: false,  
    template: '<span>Template</span>',  
    effect: 'fade',  
    effectDuration: 1000,  
    openDelay: 1000  
}).done(function(e){  
    // Search configured successfully  
}).fail(function(e){  
    // Invalid configuration  
});
```

Options

Option	Type	Description
enabled	boolean	Enable/disable Search button on screen.
hideDuringInvite	boolean	When auto-invite feature is activated, hide the Search button. When invite is dismissed, reveal the Search button again.
template	string	Custom HTML string template for Search button.
effect	string	Type of animation effect.
effectDuration	string	Type of animation effect when revealing Search button ('slide' or 'fade').
openDelay	number	Number of milliseconds before displaying Search button on screen.

Resolutions

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

open

Opens the Search Widget

Example

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

Options

Option	Type	Description
query	string	Initial question searched when window is opened.
knowledgebases	object	Array of knowledge base IDs for all further requests. Overwrites knowledgeCenterServer widget settings.
lang	string	Default language for all further requests. Overwrites knowledgeCenterServer widget settings.
media	string	Allows you to search content for media. Overwrites knowledgeCenterServer widget settings.
categories	object	Array of Category IDs for additional filter.
maxSearchResults	number	Maximum number of most relevant search results shown.
windowTitle	string	Overwrites default window title Ask a question.
hideSearchBar	boolean	Allows you to hide search input with the search button.

Resolutions

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

openDocument

Opens the Search Widget with the specified document shown.

Example

```
oMyPlugin.command('Search.openDocument').done(function(e){  
    // Document opened successfully
```

```
}).fail(function(e){  
    // Failed to open document  
});
```

Options

Option	Type	Description
documentId	string	Document ID.
knowledgeBaseId	string	Knowledge base ID of the document.
langId	string	Language ID of the document.
windowTitle	string	Overwrites default window title Ask a question

Resolutions

Status	When	Returns
resolved	When document is successfully opened	n/a
rejected	When missing mandatory arguments	'All mandatory arguments must be provided'

close

Closes the Search Widget

Example

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

Resolutions

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

showSearchButton

Makes the standalone search button visible on the screen using either the default template and CSS or customer-defined ones.

Example

```
oMyPlugin.command('Search.showSearchButton', {
    openDelay: 1000,
    duration: 1500
}).done(function(e){
    // Search shows search button successfully
}).fail(function(e){
    // Search button is already visible, side bar is active and overrides the search
    button, or search button is disabled in configuration
});
```

Options

Option	Type	Description
openDelay	number	Duration in milliseconds to delay showing the search button on the page.
duration	number	Duration in milliseconds for the show and hide animation.

Resolutions

Status	When	Returns
resolved	When the search button is enabled in the configuration, is	n/a

Status	When	Returns
	currently not visible, and the SideBar plugin is not initialized	
rejected	When the search button is not enabled in the configuration, or it's already visible, or the SideBar plugin is initialized	'Search button is already visible. Ignoring command.'
rejected	When the sidebar plugin is active the standalone search button will be disabled automatically	'SideBar is active and overrides the default search button'

hideSearchButton

Hides the standalone search button.

Example

```
oMyPlugin.command('Search.hideSearchButton', {duration: 1500}).done(function(e){  
    // Search hid search button successfully  
}).fail(function(e){  
    // Search button is already hidden  
});
```

Options

Option	Type	Description
duration	number	Duration in milliseconds for the show and hide animation.

Resolutions

Status	When	Returns
resolved	When the search button is currently visible	n/a
rejected	When the search button is already hidden	'Search button is already hidden. Ignoring command.'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

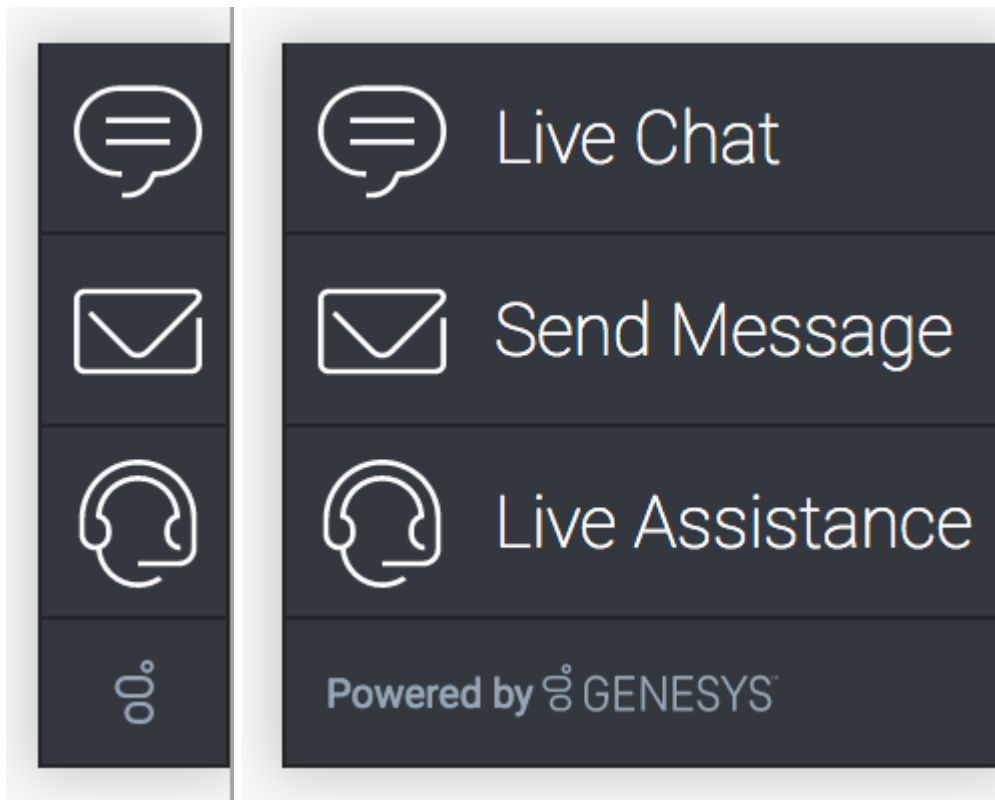
The global bus object is a debug tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see [Widgets Extensions](#) for more information about extending Genesys Widgets.

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

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

SideBar

Showing both when initially loaded on page and expanded.



- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

The Sidebar widget is displayed to the right side of the screen by default. The purpose of this Widget is to launch other Widgets with a single click. Customers can configure Widgets onto Sidebar, for which they would like to add a launch button. Sidebar Widget also supports configuring custom extension Widgets. The Sidebar UI is expanded when you hover your cursor over it, and then contracted back when you move the cursor away. Other features include configuring position, mobile support, and support adding new configuration on the fly which re-renders the sidebar.

Usage

SideBar can be launched manually by the following methods:

- Calling the **command** "SideBar.open"
- Configuring **Configuration** to show and launch custom widgets.

Dependency

The Sidebar Widget needs at-least one Widget to be configured.

Customization

All text shown in the Sidebar Widget is fully customizable and **localizable** by adding entries into your **configuration** and **localization** options.

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

Namespace

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 fullscreen templates that are optimized for both portrait and landscape orientations.

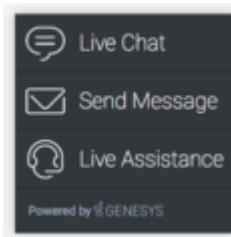
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

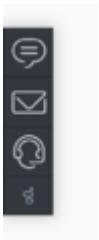
"Dark" Theme



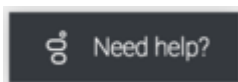
Sidebar in contracted mode -
desktop



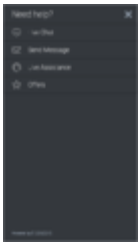
Sidebar in expanded mode -
desktop



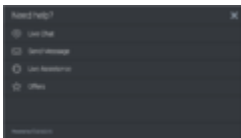
Sidebar in left side of the screen
- desktop



Sidebar launcher button in
Mobile screen.



SideBar expanded to fullscreen view in Mobile - portrait orientation

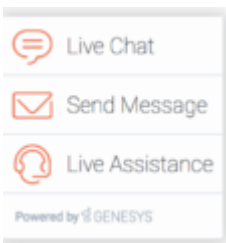


SideBar expanded to fullscreen view in Mobile - landscape orientation

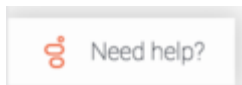
"Light" Theme



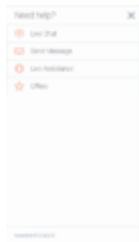
SideBar in contracted mode - desktop



SideBar in expanded mode - desktop



Sidebar launcher button in Mobile screen.



Sidebar expanded to fullscreen view in Mobile - portrait orientation



Sidebar expanded to fullscreen view in Mobile - landscape orientation

Configuration

Description

SideBar shares the configuration namespace '_genesys.widgets.sidebar'. SideBar has UI options to handle the position of sidebar on the screen, disable expand feature sidebar, hide sidebar and add new channels on the fly. The display of channels order 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',
    readyEvent: 'ChannelSelector.ready',
    clickOptions: {},

    //use your own static string or i18n query string for the below two
    displayName: 'Live Assist',
    displayTitle: 'Get live help',
    icon: 'agent'
  },
  {
    name: 'Search',
    clickCommand: 'Search.open',
    clickOptions: {},
    readyEvent: 'Search.ready',

    // Example of i18n query string: '@i18n:search.SearchName' where
    // 'search' refers to the plugin namespace and 'SearchName' refers to the property key
    // containing the actual text.
    displayName: '@i18n:search.SearchName',
    displayTitle: '@i18n:search.SearchTitle',
    icon: 'knowledge-center',
    onClick: function ($, CXBus, Common) {
      _genesys.widgets.bus.command('Search.open');
    }
  }
}

```

```

    }}
};

```

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.	right	false
expandOnHover	boolean	Enables the expand (slide-out) or contract (slide-in) behaviour of sidebar.	true	false
channels[index].name	string	Name of the channel. Used to identify official channels vs 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.	n/a	true
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	Event	Subscribes to this	n/a	false

Name	Type	Description	Default	Required
		ready event published by a plugin.		
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:<plugin namespace>.<display key>.	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:<plugin namespace>.<display key>.	n/a	false
channels[index].icon	string	Change the default icon for this channel.	n/a	false
channels[index].onClick	function	Define a custom onclick function, this overrides clickCommand and clickOptions.	n/a	false

Localization

Customer Defined Strings

For your own custom plugins, you can define string key names and values for Name and Title (tooltip) to display on sidebar. Key format has to be with Plugin name followed by 'Title' or 'Name' (for example, '< custom plugin name >Title'). As a case in point, a plugin named 'MyPlugin' will have 'MyPluginName' and 'MyPluginTitle' as keys.

Important

For information on how to setup localization, please refer to the [Localization Guide](#)

Strings

```
{
  "SidebarTitle": "Need help?",
  "ChannelSelectorName": "Live Assistance",
  "ChannelSelectorTitle": "Get assistance from one of our agents right away",
  "SearchName": "Search",
  "SearchTitle": "Search",
  "OffersName": "Offers",
  "OffersTitle": "Offers",
  "CallUsName": "Call Us",
  "CallUsTitle": "Call Us details",
  "CallbackName": "Callback",
  "CallbackTitle": "Receive a Call",
  "SendMessageName": "Send Message",
  "SendMessageTitle": "Send Message",
  "WebChatName": "Live Chat",
  "WebChatTitle": "Live Chat"
}
```


API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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 atleast 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. 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:<plugin namespace>.<display key>.
channels[index].displayTitle	string or i18n query string	Change the default tooltip content for this channel with

Option	Type	Description
		your own static string or to achieve localization, use i18n query string. Syntax: @i18n:<plugin namespace>.<display key>.
channels[index].readyEvent	string	Subscribes to this ready event published by a plugin.
channels[index].icon	string	Change the default Icon for this channel.
channels[index].onClick	function	Define a custom onclick function, this overrides clickCommand and clickOptions.

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, 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, it slides out from the sides of the screen in desktop but expands to full screen in mobiles.

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

Slides back showing only the channel buttons in desktop and sidebar launcher button in mobile.

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

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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 all channel details in desktops.	n/a
contracted	Sidebar widget has slid back to initial state showing minimal channel details in desktops.	n/a
mobile.expanded	Sidebar widget has expanded to full screen in mobiles, showing all channel details.	n/a
mobile.contracted	Sidebar widget is minimized back to initial state in mobiles.	n/a

StatsService

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

StatsService exposes a high-level API for utilizing Genesys Stats services. You can use these services to fetch estimated waiting time details for each channel like Chat, Callus, etc. and display it across the channels.

Usage

StatsService and Channel Selector widget works together right out of the box and display the Estimated Waiting Time details across the channels. Using Channel Selector uses StatsService.

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

Namespace

Stats Service plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
Configuration	stats
CXBus - API Commands & API Events	StatsService

Customization

StatsService has no customization options. It is meant as a plug-n-play type of plugin and works as-is.

Configuration

Description

StatsService share the configuration namespace '_genesys.cxwidget.stats'. StatsService has connection settings to fetch EWT details from each channel.

Example

```
window._genesys.cxwidget.stats = {  
  ajaxTimeout: 3000,  
  ewt: {  
    dataURL: 'http://10.0.0.121:7777/genesys/1/service/ewt-for-vq',  
    apikey: 'n3eNkgLLgLKXREBMjGm6lyg0HHOK8VA'  
  }  
};
```

Options

Name	Type	Description	Default	Required
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout	3000	n/a
ewt.apikey	string	Apigee Proxy secure token	n/a	Yes, if using Apigee Proxy
ewt.dataURL	URL String	URL of GMS server	n/a	Always

Localization

No Localization options

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('StatsService.getUserAgent');
```

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('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	Apigee Proxy secure token
ewt.dataURL	URL String	URL of GMS server
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout.

Resolutions

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

getStats

Make a request to Genesys Stats server to fetch EWT details.

Example

```
oMyPlugin.command('StatsService.getStats', {
    group: 'EWT',
    vqname: 'chat_ewt_test_eservices'
}).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 like EWT, etc.

Option	Type	Description
vqname	string	Mention virtual queue name for EWT.

Resolutions

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

getUserAgent

Receive User Agent data from the browser

Example

```
oMyPlugin.command('StatsService.getUserAgent').done(function(e){  
    // StatsService got user agent data successfully  
    // e == Object of User agent data  
}).fail(function(e){  
    // StatsService failed to get user agent data  
});
```

Resolutions

Status	When	Returns
resolved	Always	UAParser object with functions to find browser (.getBrowser()), OS (.getOS()), mobile/web detection (.mobileCheck())

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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: {(EWT AJAX Response data)}}}
error.ewt	An error occurred between the client and the server for EWT	{{(AJAX data Response)}}

Toaster

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

The Toaster plugin provides a toast view control that widgets can inject their UI into. The Toaster plugin accepts an HTML UI and puts it inside a toast view and displays the UI onscreen in the lower-bottom-right of the screen. When it is opened it will slide up from the bottom. When it is closed it will slide down until it is offscreen.

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 very 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).

Important

Only one toast can be shown at a time. If you attempt to open a second toast, the first toast will be dismissed automatically before showing the second toast.

Namespace

Toaster plugin has the following namespaces tied-up with each of the following types.

Type	Namespace
CXBus - API Commands & API Events	Toaster
CSS	.cx-toaster

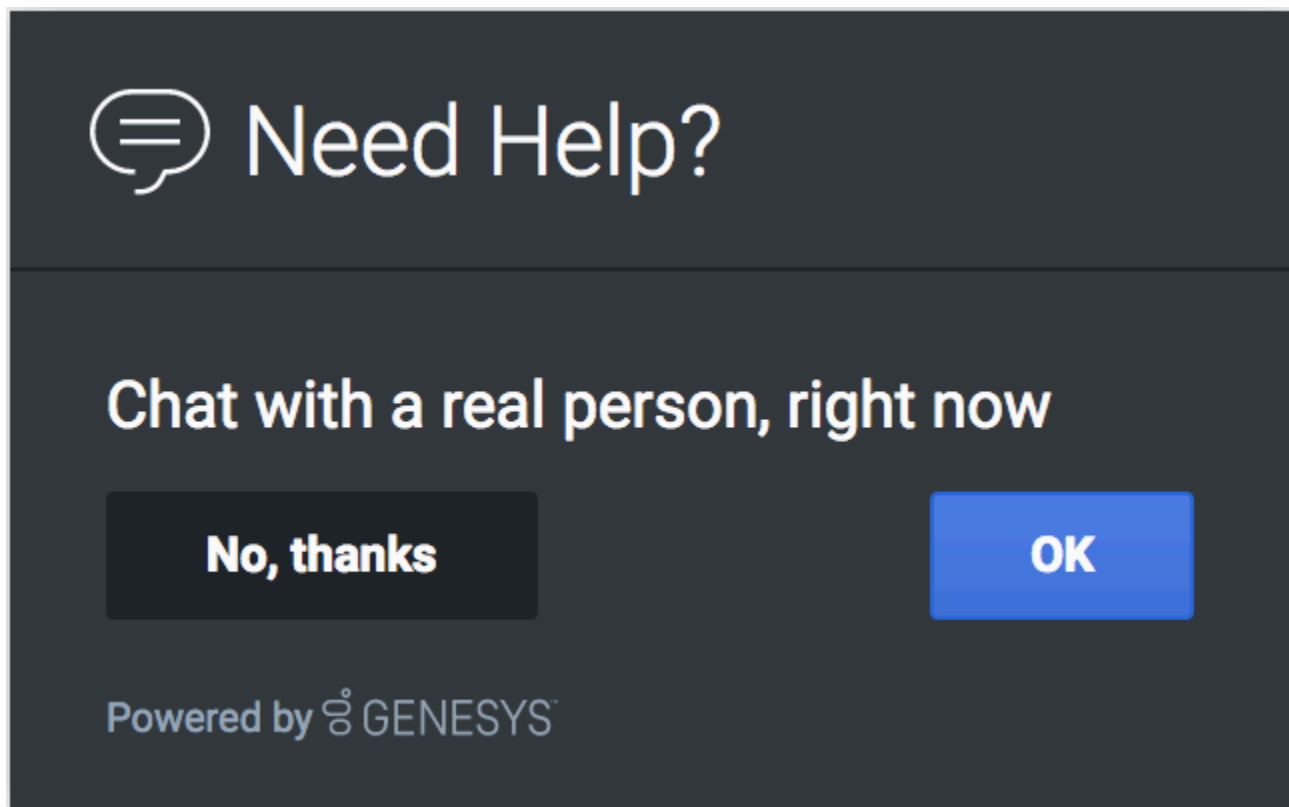
Customization

Toaster does not have customization options.

Mobile Support

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

Screenshots



Configuration

No configuration options.

Localization

No localization options.

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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'  
  }  
}).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	When Toaster is successfully opened	n/a
rejected	When no toaster type is specified	'No content was provided. Toaster has ignored your command'
rejected	When 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	When toaster is successfully closed.	n/a
rejected	When Toaster is already closed.	'Toaster view is already closed'
rejected	When Toaster view is immutable.	'Toaster view is currently reserved'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('Toaster.ready', function(e){});
```

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

WindowManager

- [Configuration](#)
- [Localization](#)
- [API Commands](#)
- [API Events](#)

Overview

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

One group type is "Dock View". Both WebChat and SendMessage utilize this group to show their toast-like UI docked in the lower-bottom-right of the screen. This group automatically arranges the two widgets stacked horizontally and when one widget closes, the stack collapses towards the right. Widgets can register themselves into these WindowManager groups and let it do all the work.

Another group type is "Side Button". WebChat and SendMessage also utilize this group to show their launcher buttons 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 will collapse 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.

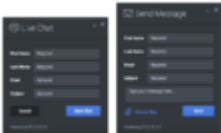
Customization

Toaster does not have customization options.

Screenshots



- Side button group



- Dock view group

Configuration

No configuration options.

Localization

No localization options.

API Commands

Once you've registered your own plugin on the bus, you can call commands on other registered plugins. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('WindowManager.registerDockView', {html: '<div>HTML</div>'});
```

registerDockView

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

Example

```
oMyPlugin.command('WindowManager.registerDockView', {html: '  
<div>Template</div>' }).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	When the html template is successfully opened and registered in dock view	n/a
rejected	When 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: '<div>Button  
Text</div>'}).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	When the html button is successfully registered	n/a
rejected	When no html template is found	'No button template found to register'

API Events

Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

Important

The global bus object is a debug 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('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)}