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 Current

6/17/2022

Table of Contents

Genesys Widgets Reference

> **Tip**
> The latest version of our documentation (titled "**Current**") relates to release **9.0.x**.

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

WebChatService

Overview

WebChatService exposes a high-level API for utilizing Genesys chat services. You can you 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.

Limitations

Multiple instances of the same chat session

After starting a chat session, that session can be opened in any number of new tabs on the same site. Each tab runs an independent instance of WebChat connected to the same chat session. Instances are not synchronized with each other, however, which may result in unusual behavior. The following limitations apply:

- When using CometD mode (long-polling or websockets), messages will not be synchronized between tabs.
- Inactivity messages and other dialog messages will not be synchronized between tabs.
- The number of unread messages displayed will not be synchronized between tabs.
- The minimized state of WebChat will not be synchronized between tabs.

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: 'n3eNkgxxxxxxxxxxx8VA',
        dataURL: 'https://api.genesyscloud.com/gms-chat/2/chat',
        enableCustomHeader: true,

        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
        },

        async: {

                enabled: true,

                getSessionData: function(sessionData, Cookie, CookieOptions) {
```

```
                        // Note: You don't have to use Cookies. You can, instead, store in a
secured location like a database.

                        Cookie.set('customer-defined-session-cookie',
JSON.stringify(sessionData), CookieOptions);
                },

                setSessionData: function(Open, Cookie, CookieOptions) {

                        // Retrieve from your secured location.

                        return Cookie.get('customer-defined-session-cookie');
                }
        }
};
```

Options

Name	Type	Description	Default	Required	Introduced / Updated
apikey	string	Apigee Proxy secure token. `Note: This option is only supported in GMS REST mode.`	n/a	Yes, if using Apigee Proxy	
endpoint	string	Manually select the endpoint to initiate chat on.	n/a	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	
enableCustomHeader	boolean	Enables the use of the custom authorization header defined in _genesys.widgets.basic.header static config. Attaches the custom authorization header to all WebChatService	false	No	9.0.002.06

Name	Type	Description	Default	Required	Introduced / Updated
		request.			
cometD	object	Object container for CometD configuration options.	{enabled: false, cometURL: , *channel: '/service/chatV2/customer-support', apiURL:* , 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 through websockets. If set to false,	true	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
		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'	n/a	
userData	object	Arbitrary attached data to include when initiating a chat.	{}	n/a	
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout.	3000	n/a	
xhrFields	object	Allows you to set the properties for the AJAX xhrFields object (for example, {withCredentials: false}). `Note: This option is only supported in GMS REST mode.`	{withCredentials: false}	n/a	
pollExceptionLimit	number	Number of successive poll exceptions (chat server offline) before WebChatService publishes 'chatServerWentOffline'.	5	n/a	
restoreTimeout	number	Number of milliseconds before restore timeout.	60000		

Name	Type	Description	Default	Required	Introduced / Updated
		Prevents the chat session from restoring after a certain time away from the session (for example, user navigated to a different site during chat and never ended the session).			
async	object	Object container for Async mode configuration options.	{enabled: false}	No	9.0.002.06
async.enabled	boolean	Enable Asynchronous Chat where a chat session can be active indefinitely. When you close WebChat without ending the chat session, the session will simply go dormant. When you open WebChat again, the session will restore and continue chatting where left off. Currently, Async Chat is supported only in cometD mode and it should be enabled.	false	n/a	9.0.002.06
async.getSessionData	function	A function that you can define to retrieve updated session data	none	Yes, when Async WebChat mode is enabled	9.0.002.06

Name	Type	Description	Default	Required	Introduced / Updated
		from WebChatService plugin. This function is called back when starting a new Async chat session for the first time or when the sessionData changes over the course of an active chat session. This function takes the following arguments - sessionData (current active session data), Cookie (Widgets Internal cookie reference) and CookieOptions (a parameter that is needed when using Widgets Cookie). The purpose of this function is to provide you the active session data so that it can be stored somewhere safe and secure. Later this needs to be provided in the below setSessionData function to restore the chat session. Refer to the example for usage.			
async.setSessionData	function	A function that you can define	none	Yes, when Async WebChat	9.0.002.06

Name	Type	Description	Default	Required	Introduced / Updated
		to return the session data to WebChatService plugin. During initialization, WebChatService plugin will call this function to check if any session data is returned. If found, WebChatService tries to restore the chat session using this session data and open WebChat Widget. WebChatService will also pass the following arguments into this function - Open (WebChat current open state value), Cookie (Widgets Internal cookie reference) and CookieOptions (a parameter that will be needed when using Widgets Cookie). Refer to the example for usage.		mode is enabled	

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 data object 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', {message: 'hi'}).done(function(e){

        // WebChatService sent typing successfully

}).fail(function(e){

        // WebChatService failed to send typing
});
```

Options

Option	Type	Description
message	string	The message you want to send along with the typing notification.

Resolutions

Status	When	Returns
resolved	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 object 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
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, except when using Async mode.

Example

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

        // WebChatService restored successfully

}).fail(function(e){

        // WebChatService failed to restore
});
```

Options

Option	Type	Description	Accepted Values	Introduced / Updated
sessionData	object	Applicable when using Async mode only. The session data that is needed to restore the WebChat in Async mode. It is a Key value pair object containing the values mentioned.	secureKey, userId, alias, sessionID	9.0.002.06

Resolutions

Status	When	Returns	Introduced / Updated
resolved	Session has been found.	n/a	

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

getTranscript

Fetch an array of all messages in the chat session.

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

Example

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

        // WebChatService got transcript successfully
        // e == Object with an array of messages

}).fail(function(e){

        // WebChatService failed to get transcript
});
```

Resolutions

Status	When	Returns
resolved	Always	Object with an array of messages

getAgents

Return a list of agents 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), 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)

Status	When	Returns
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', fileName:
'myfile.txt'}).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
fileName	string	This is the name of the file to be downloaded from the session. It is an optional field.

Resolutions

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

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

getSessionData

[Introduced: 9.0.002.06]

To retrieve the active session data at any time.

Example

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

Resolutions

Status	When	Returns
resolved	Always	{secureKey: (string), sessionID: (number/string), alias: (number/string), userId: (number/string)}
rejected	Never	undefined

fetchHistory

[Introduced: 9.0.002.06]

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

Example

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

Resolutions

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

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 object, allowing you to manipulate the values before it is rendered in the transcript.

Example

```
oMyPlugin.command('WebChatService.registerPreProcessor', {preprocessor: function(message){

        message.text = message.text + ' some preprocessing text';

        return message;

}}).done(function(e){

        // WebChatService registered preprocessor function
        // e == function that was registered

}).fail(function(e){

        // WebChatService failed to register function

});
```

Options

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

Resolutions

Status	When	Returns
resolved	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>'.

verifySession

Checks for existing WebChat session before triggering a proactive invite.

Example

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

Resolutions

Status	When	Returns
resolved	A session exists or not	A boolean 'sessionActive' which holds the session state.

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){});
```

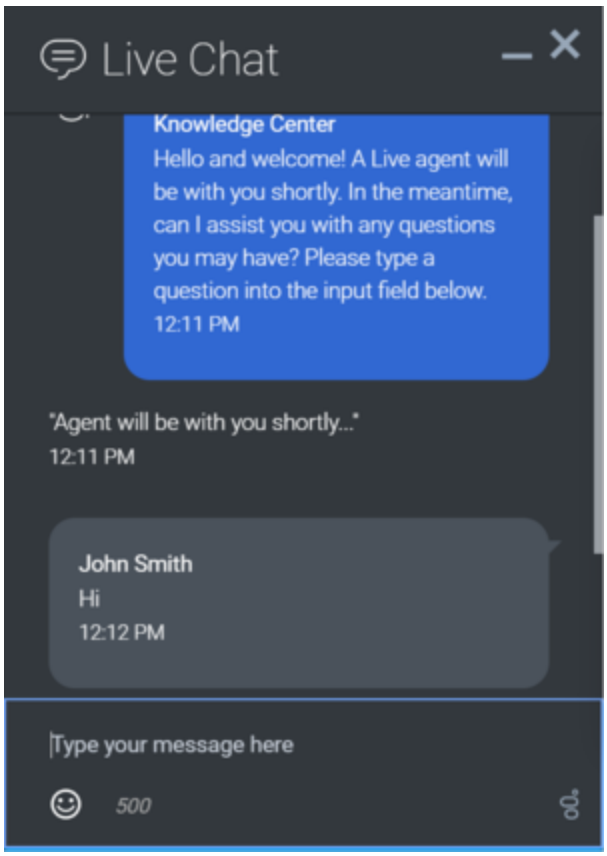
Name	Description	Data	Introduced / Updated
ready	WebChatService is initialized and ready to accept commands.	n/a	
restored	Chat session has been restored after page navigation or refresh. In Asynchronous mode, this event includes data indicating whether a chat session has been restored in Async mode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

Name	Description	Data	Introduced / Updated
	through. When chat server is comes back online, 'chatServerBackOnline' is published.		
messageReceived	A new message has been received from the server. Includes text messages, status messages, notices, and other message types.	{originalMessages: (object), messages: (array of objects), restoring: (boolean), sessionData: (object)}	9.0.002.06
error	An error occurred between the client and the server.	(AJAX Response)	
started	Chat session has successfully started.	(AJAX Response containing session data)	
ended	Chat session has successfully ended.	n/a	
agentTypingStarted	Agent has started typing a new message.	(AJAX Response)	
agentTypingStopped	Agent has stopped typing.	(AJAX Response)	
pollingStarted	Chat server automatic polling has started.	n/a	
pollingStopped	Chat server automatic polling has stopped.	n/a	
clientConnected	Indicates the user has been connected to the chat session.	{message: (object), agents: (object), numAgentsConnected: (number)}	
clientDisconnected	Indicates the user has been disconnected from 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:	

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

Name	Description	Data	Introduced / Updated
	events is to allow for separate handling of both scenarios. **Important** This event is applicable only when using WebChat with GMS API.		
chatServerBackOnline	Chat server has come back online after going offline. This will only be published after 'chatServerWentOffline'. **Important** This event is applicable only when using WebChat with GMS API.	n/a	
connectionPending	If there is a connection problem and WebChatService is trying to reconnect, this event will be published. Published before 'chatServerWentOffline'. **Important** This event is applicable only when using WebChat with GMS API.	n/a	
connectionRestored	Is published when the connection has been reestablished. Publishes at the same time as 'chatServerBackOnline'.	n/a	

WebChat



- 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.mychatservice", 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, the 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 is 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

The 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 full-screen templates that are optimized for both portrait and landscape orientations.

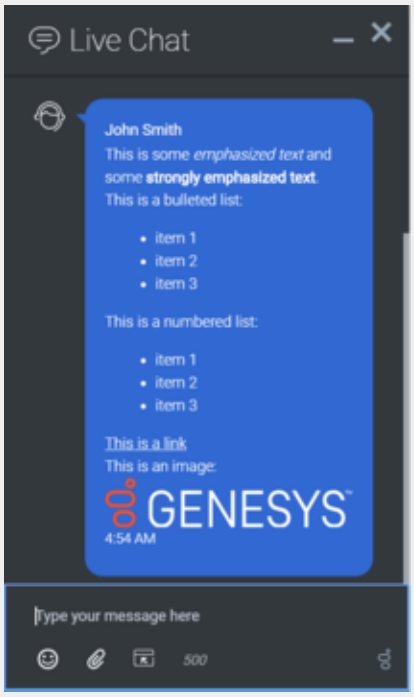
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Markdown Support

Starting in Genesys Widgets version 9.0.013.04, Markdown text formatting is supported in WebChat. Our implementation follows the CommonMark spec. Please review the syntax rules and available formatting.

Example

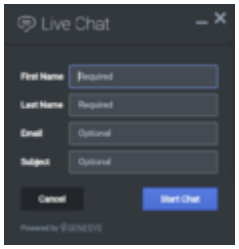
Dark Theme	`# This is a title!`
	
	`This is some *emphasized text* and some **strongly emphasized text**`
		Light Theme



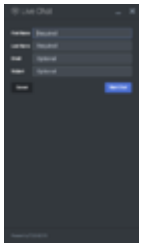

```
This is a bulleted list:

* item 1
* item 2
* item 3

This is a numbered list:

1. item 1
2. item 2
3. item 3

[This is a
link](http://www.genesys.com)

This is an image:

![Alt
Text](https://genbin.genesys.com/
media/
genesys_logo_tagline.svg)
```

Screenshots

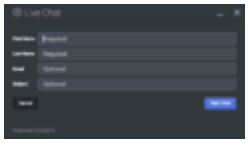
"Dark" Theme

- 

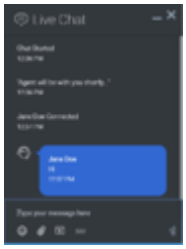
  Desktop docked view showing form

- 

Mobile fullscreen view in portrait orientation showing form

- 

Mobile fullscreen view in Landscape orientation showing form

- 

Desktop docked view showing transcript

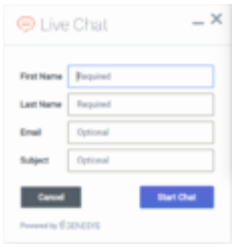
- 

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

Important

You may choose to apply different colors/themes to your widgets, and you can visit Styling the Widgets to find out how.

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: 'n3eNkgLLgLKXREBMYjGm6lygOHHOK8VA',
        dataURL: 'https://api.genesyscloud.com/gms-chat/2/chat',
        userData: {},
        emojis: true,
        uploadsEnabled: false,
        confirmFormCloseEnabled: true,
        actionsMenu: true,
        maxMessageLength: 140,

        autoInvite: {

                enabled: false,
                timeToInviteSeconds: 10,
                inviteTimeoutSeconds: 30
        },

        chatButton: {

                enabled: true,
                template: '<div class="cx-widget cx-webchat-chat-button cx-side-button"
role="button" tabindex="0" data-message="ChatButton" data-gcb-service-node="true"><span
class="cx-icon" data-icon="chat"></span><span class="i18n cx-chat-button-label" data-
message="ChatButton"></span></div>',
                effect: 'fade',
                openDelay: 1000,
                effectDuration: 300,
                hideDuringInvite: true
        },

        async: {

                enabled: true,
                newMessageRestoreState: 'minimized',

                getSessionData: function(sessionData, Cookie, CookieOptions) {

                        // Note: You don't have to use Cookies. You can, instead, store in a
secured location like a database.

                        Cookie.set('customer-defined-session-cookie',
JSON.stringify(sessionData), CookieOptions);
```

```
                },
                setSessionData: function(Open, Cookie, CookieOptions) {
                        // Retrieve from your secured location.
                        return Cookie.get('customer-defined-session-cookie');
                }
        },
        minimizeOnMobileRestore: false,
        markdown: false,
        ariaIdleAlertIntervals:[50,25,10],
        ariaCharRemainingIntervals:[75, 25, 10]
};
```

Options

Name	Type	Description	Default	Required	Introduced / Updated
emojis	boolean	Enable/disable Emoji menu inside chat message input. Emojis are supported using unicode characters and the list includes ☺ U+263A (smile), 👍 U+1F44D (thumbs up) and ☹ U+2639 (sad).	false	n/a	
form	object	An object containing a custom registration form definition. The definition placed here becomes the default registration form layout for WebChat. See Customizable Chat Registration Form.	A basic registration form is defined internally by default	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
uploadsEnabled	boolean	Show/Hide the Send File button. The button will be shown if the value is set to true.	false	n/a	
confirmFormCloseEnabled	boolean	Enable or disable displaying a confirmation message before closing WebChat if information has been entered into the registration form.	true	n/a	
timeFormat	number/string	This sets the time format for the timestamps in this widget. It can be 12 or 24.	12	false	
actionsMenu	boolean	Enable/disable actions menu next to chat message input.	true	n/a	
maxMessageLength	number	Set a character limit that the user can input into the message area during a chat. When max is reached, user cannot type any more.	500	n/a	
charCountEnabled	boolean	Show/Hide the number of characters remaining in the input message area while the user is typing.	false	n/a	
autoInvite.enabled	boolean	Enable/disable auto-invite feature. Automatically	false	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
		invites user to chat after user idles on page for preset time. **Important** In case of running Widgets in lazy load mode, this option requires WebChat plugin to be pre-loaded.			
autoInvite.timeToInviteSeconds	number	Number of seconds of idle time before inviting customer to chat.	5	n/a	
autoInvite.inviteTimeoutSeconds	number	Number of seconds to wait, after showing invite, before closing chat invite. **Important** When the focus is on the Invite window, the chat invite will not auto close upon the specified timeout. In this scenario, you must click the Close button to manually close the Invite window. This is to support the logical and predictable focus order as recommended by WCAG 2.4.3:Focus Order.	30	n/a	
chatButton.enabled	boolean	Enable/disable chat button on screen. **Important** In case of running Widgets	false	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
		in lazy load mode, this option requires WebChat plugin to be pre-loaded.			
chatButton.template	string	Custom HTML string template for chat button.	`<div class="cx-widget cx-webchat-chat-button cx-side-button" role="button" tabindex="0" data-message="ChatButton" data-gcb-service-node="true"><span class="cx-icon" data-icon="chat"></span><span class="i18n cx-chat-button-label" data-message="ChatButton"></span></div>`	n/a	
chatButton.effect	string	Type of animation effect when revealing chat button. 'slide' or 'fade'.	fade	n/a	
chatButton.openDelay	number	Number of milliseconds before displaying chat button on screen.	1000	n/a	
chatButton.effectDuration	number	Length of animation effect in milliseconds.	300	n/a	
chatButton.hideDuringInvite	boolean	When auto-invite feature is activated, hide the chat button. When invite is dismissed, reveal the chat button again.	true	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
async	object	Object container for Async mode configuration options.	{enabled: false, newMessageRestoreState: 'full'}	Yes, if using Async mode	9.0.002.06
async.enabled	boolean	Enable Asynchronous Chat where a chat session can be active indefinitely. When you close WebChat without ending the chat session, the session will simply go dormant. When you open WebChat again, the session will restore and continue chatting where left off. Currently, Async Chat is supported only in cometD mode and it should be enabled.	false	n/a	9.0.002.06
async.newMessageRestoreState	string	Determines how WebChat should be displayed when a new message arrives if WebChat is closed. Accepted values are 'full' and 'minimized'. 'full' means WebChat appears on screen displaying new messages in the transcript area as a new	full	n/a	9.0.002.06

Name	Type	Description	Default	Required	Introduced / Updated
		message arrives. 'minimized' means WebChat opens in a minimized state along with a counter in the title bar area indicating how many new messages are present.			
async.getSessionData	function	A function that you can define to retrieve updated session data from WebChatService plugin over the course of an active chat session. This function takes the following arguments - sessionData (current active session data), Cookie (Widgets Internal cookie reference) and CookieOptions (a parameter that is needed when using Widgets Cookie). The purpose of this function is to provide you the active session data so that it can be stored somewhere safe and secure. Later this needs to be provided in the below	none	Yes, when Async WebChat mode is enabled	9.0.002.06

Name	Type	Description	Default	Required	Introduced / Updated
		setSessionData function to restore the chat session. Refer to the example for usage.			
async.setSessionData	function	A function that you can define to return the session data to WebChat plugin. During initialization, WebChatService plugin will call this function to check if any session data is returned. If found, WebChatService tries to restore the chat session using this session data and open WebChat Widget. WebChatService will also pass the following arguments into this function - Open (WebChat current open state value), Cookie (Widgets Internal cookie reference) and CookieOptions (a parameter that will be needed when using Widgets Cookie). Refer to the example for usage.	none	Yes, when Async WebChat mode is enabled	9.0.002.06
minimizeOnMobileRestore	Boolean	Enable/disable the minimized state of webchat on	false	n/a	9.0.013.02

Name	Type	Description	Default	Required	Introduced / Updated
		chat restore. **Important** This option is only for mobile mode.			
markdown	boolean	Enable/disable the markdown feature for chat messages.	false	n/a	9.0.014.02
ariaIdleAlertIntervals	array/boolean	An array containing the intervals as a percentage at which the screen reader will announce the remaining idle time. By default, it is enabled with the following time intervals, and it is customizable according to the user's needs. Configuring a value of 'false' will let the screen reader call out idle time for every change.	[100, 75, 50, 25, 10]	n/a	9.0.016.11
ariaCharRemainingIntervals	array/boolean	An array containing the intervals as a percentage at which the screen reader will announce the remaining characters when the user inputs text into the message area. By default, it is enabled with the following intervals, and it is customizable	[50, 25, 10]	n/a	9.0.016.11

Name	Type	Description	Default	Required	Introduced / Updated
		according to the user needs. Configuring a value of 'false' will let the screen reader call out remaining characters for every change.			
metaDataEnabled	boolean	Enable or disable WebChat MetaData.	true	n/a	9.0.017.26
enableUrlTrailingSlash	boolean	Enable or disable trailing slash at the end of the dataURL when the start chat connection request is sent to the server. **Important** This option is applicable for REST mode only.	true	n/a	9.0.017.28

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 set up 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",
                        "BotNameDefault": "Bot",
                        "BotConnected": "<%Bot%> Connected",
                        "BotDisconnected": "<%Bot%> Disconnected",
                        "SupervisorNameDefault": "Supervisor",
                        "SupervisorConnected": "<%Agent%> Connected",
                        "SupervisorDisconnected": "<%Agent%> Disconnected",
                        "AgentTyping": "...",
                        "AriaAgentTyping": "Agent is typing",
                        "AgentUnavailable": "Sorry. There are no agents available. Please try
later.",
                        "ChatTitle": "Live Chat",
                        "ChatEnd": "X",
                        "ChatClose": "X",
```

```
                        "ChatMinimize": "Min",
                        "ChatFormFirstName": "First Name",
                        "ChatFormLastName": "Last Name",
                        "ChatFormNickname": "Nickname",
                        "ChatFormEmail": "Email",
                        "ChatFormSubject": "Subject",
                        "ChatFormPlaceholderFirstName": "Required",
                        "ChatFormPlaceholderLastName": "Required",
                        "ChatFormPlaceholderNickname": "Optional",
                        "ChatFormPlaceholderEmail": "Optional",
                        "ChatFormPlaceholderSubject": "Optional",
                        "ChatFormSubmit": "Start Chat",
                        "AriaChatFormSubmit": "Start Chat",
                        "ChatFormCancel": "Cancel",
                        "AriaChatFormCancel": "Cancel",
                        "ChatFormClose": "Close",
                        "ChatInputPlaceholder": "Type your message here",
                        "ChatInputSend": "Send",
                        "AriaChatInputSend": "Send",
                        "ChatEndQuestion": "Are you sure you want to end this chat session?",
                        "ChatEndCancel": "Cancel",
                        "ChatEndConfirm": "End chat",
                        "AriaChatEndCancel": "Cancel",
                        "AriaChatEndConfirm": "End chat",
                        "ConfirmCloseWindow": "Are you sure you want to close chat?",
                        "ConfirmCloseCancel": "Cancel",
                        "ConfirmCloseConfirm": "Close",
                        "AriaConfirmCloseCancel": "Cancel",
                        "AriaConfirmCloseConfirm": "Close",
                        "ActionsDownload": "Download transcript",
                        "ActionsEmail": "Email transcript",
                        "ActionsPrint": "Print transcript",
                        "ActionsCobrowseStart": "Start Co-browse",
                        "AriaActionsCobrowseStartTitle": "Opens the Co-browse session",
                        "ActionsSendFile": "Attach Files",
                        "AriaActionsSendFileTitle": "Opens a file upload dialog",
                        "ActionsEmoji": "Send Emoji",
                        "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": "Share 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",
                        "AriaInviteAccept": "Start chat",
                        "AriaInviteReject": "No thanks",
                        "ChatError": "There was a problem starting the chat session. Please
retry.",
                        "ChatErrorButton": "OK",
                        "AriaChatErrorButton": "OK",
                        "ChatErrorPrimaryButton": "Yes",
                        "ChatErrorDefaultButton": "No",
                        "AriaChatErrorPrimaryButton": "Yes",
                        "AriaChatErrorDefaultButton": "No",
                        "DownloadButton": "Download",
                        "AriaDownloadButton": "Download",
                        "FileSent": "has sent:",
                        "FileTransferRetry": "Retry",
```

```
                        "AriaFileTransferRetry": "Retry",
                        "FileTransferError": "OK",
                        "AriaFileTransferError": "OK",
                        "FileTransferCancel": "Cancel",
                        "AriaFileTransferCancel": "Cancel",
                        "RestoreTimeoutTitle": "Chat ended",
                        "RestoreTimeoutBody": "Your previous chat session has timed out.
Would you like to start a new one?",
                        "RestoreTimeoutReject": "No thanks",
                        "RestoreTimeoutAccept": "Start chat",
                        "AriaRestoreTimeoutAccept": "Start chat",
                        "AriaRestoreTimeoutReject": "No thanks",
                        "EndConfirmBody": "Would you really like to end your chat session?",
                        "EndConfirmAccept": "End chat",
                        "EndConfirmReject": "Cancel",
                        "AriaEndConfirmAccept": "End chat",
                        "AriaEndConfirmReject": "Cancel",
                        "SurveyOfferQuestion": "Would you like to participate in a survey?",
                        "ShowSurveyAccept": "Yes",
                        "ShowSurveyReject": "No",
                        "AriaShowSurveyAccept": "Yes",
                        "AriaShowSurveyReject": "No",
                        "UnreadMessagesTitle": "unread",
                        "AriaYouSaid": "You said",
                        "AriaSaid": "said",
                        "AriaSystemSaid": "System said",
                        "AriaWindowLabel": "Live Chat Window",
                        "AriaMinimize": "Live Chat Minimize",
                        "AriaMaximize": "Live Chat Maximize",
                        "AriaClose": "Live Chat Close",
                        "AriaEmojiStatusOpen": "Emoji picker dialog is opened",
                        "AriaEmojiStatusClose": "Emoji picker dialog is closed",
                        "AriaEmoji": "emoji",
                        "AriaCharRemaining": "Characters remaining",
                        "AriaMessageInput": "Message box",
                        "AsyncChatEnd": "End Chat",
                        "AsyncChatClose": "Close Window",
                        "AriaAsyncChatEnd": "End Chat",
                        "AriaAsyncChatClose": "Close Window",
                        "DayLabels": [
                                "Sun",
                                "Mon",
                                "Tue",
                                "Wed",
                                "Thur",
                                "Fri",
                                "Sat"
                        ],
                        "MonthLabels": [
                                "Jan",
                                "Feb",
                                "Mar",
                                "Apr",
                                "May",
                                "Jun",
                                "Jul",
                                "Aug",
                                "Sept",
                                "Oct",
                                "Nov",
                                "Dec"
                        ],
                        "todayLabel": "Today",
```

```
                        "Errors": {
                                "102": "First name is required.",
                                "103": "Last name is required.",
                                "161": "Please enter your name.",
                                "201": "The file could not be sent.<br/><strong><p
class='filename' title='<%FilenameFull%>'>'<%FilenameTruncated%>'</p></strong><p class='cx-
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='cx-
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='cx-
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.",
                                "401": "We're sorry but we are not able to authorize the chat
session. Would you like to start a new chat?",
                                "404": "We're sorry but we cannot find your previous chat
session. Would you like to start a new chat?",
                                "500": "We're sorry, an unexpected error occurred with the
service. Would you like to close and start a new Chat?",
                                "503": "We're sorry, the service is currently unavailable or
busy. Would you like to close and start a new Chat again?",
                                "ChatUnavailable": "We're sorry but we cannot start a new
chat at this time. Please try again later.",
                                "CriticalFault": "Your chat session has ended unexpectedly
due to an unknown issue. We apologize for the inconvenience.",
                                "StartFailed": "There was an issue starting your chat
session. Please verify your connection and that you submitted all required information
properly, then try again.",
                                "MessageFailed": "Your message was not received successfully.
Please try again.",
                                "RestoreFailed": "We're sorry but we were unable to restore
your chat session due to an unknown error.",
                                "TransferFailed": "Unable to transfer chat at this time.
Please try again later.",
                                "FileTransferSizeError": "The file could not be
sent.<br/><strong><p class='filename'
title='<%FilenameFull%>'>'<%FilenameTruncated%>'</p></strong><p class='cx-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='cx-
advice'>There was an unexpected disconnection. Try again?</p>",
                                "Generic": "<div style='text-align:center'>An unexpected
error occurred.</div>",
                                "pureengage-v3-rest-INVALID_FILE_TYPE": "Invalid file type.
Only Images are allowed.",
                                "pureengage-v3-rest-LIMIT_FILE_SIZE": "File size is larger
than the allowed size.",
```

```
                                "pureengage-v3-rest-LIMIT_FILE_COUNT": "The maximum number of
attached files exceeded the limit.",
                                "pureengage-v3-rest-INVALID_CONTACT_CENTER": "Invalid x-api-
key transport configuration.",
                                "pureengage-v3-rest-INVALID_ENDPOINT": "Invalid endpoint
transport configuration.",
                                "pureengage-v3-rest-INVALID_NICKNAME": "First Name is
required.",
                                "pureengage-v3-rest-AUTHENTICATION_REQUIRED": "We're sorry
but we are not able to authorize the chat session.",
                                "purecloud-v2-sockets-400": "Sorry, something went wrong.
Please verify your connection and that you submitted all required information properly, then
try again.",
                                "purecloud-v2-sockets-500": "We're are sorry, an unexpected
error occurred with the service.",
                                "purecloud-v2-sockets-503": "We're sorry, the service is
currently unavailable."
                        }
                }
        }
}
```

API Commands

Once you've registered your 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.

open

Opens the WebChat UI.

Example

```
oMyPlugin.command('WebChat.open', {

        userData: {},
        form: {

                autoSubmit: false,
                firstname: 'John',
                lastname: 'Smith',
                email: 'John@mail.com',
                subject: 'Customer Satisfaction'
        },
        formJSON: {...},
        markdown: false
```

```
}).done(function(e){

        // WebChat opened successfully

}).fail(function(e){

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

Options

Option	Type	Description	Introduced / Updated
form	object	Object containing form data to prefill in the chat entry form and optionally auto-submit the form.	
form.autoSubmit	boolean	Automatically submit the form. Useful for bypassing the entry form step.	
form.firstname	string	Value for the first name entry field.	
form.lastname	string	Value for the last name entry field.	
form.email	string	Value for the email entry field.	
form.subject	string	Value for the subject entry field.	
formJSON	object	An object containing a custom registration form definition. See Customizable Chat Registration Form.	
userData	object	Object containing arbitrary data that gets sent to the server. Overrides userData set in the webchat configuration object.	
async	boolean	Starts a new chat either in asynchronous or normal mode based on the boolean value. Note that unless async static configuration is defined, a chat in normal mode will start automatically.	9.0.002.06
markdown	boolean	The markdown feature	9.0.014.02

Option	Type	Description	Introduced / Updated
		for chat messages.	
id	string	A Unique identifier of a chat session that helps to identify the instance of that session and its associated events. A random value is automatically generated and assigned when no value is passed explicitly.	9.0.014.09

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 = uniminimized.

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	Accepted Values
type	string	Switch the rendering type of the injected message between text and html.	text, html
name	string	Specify a name label for the message to identify what service or widget has injected the message.	n/a
text	string	The content of the message. Either plain text or HTML.	n/a
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.	true, false
bubble.fill	string of valid CSS color value	The content of the message. Either plain text or HTML.	n/a
bubble.radius	string of valid CSS border radius vale	The border radius you'd like for the bubble.	n/a
bubble.time	boolean	If you'd like to show the timestamp for the bubble.	true, false
bubble.name	boolean	If you'd like to show the name for the bubble.	true, false
bubble.direction	string	Which direction you want the message bubble to come from.	left, right, none
bubble.avatar.custom	string or HTML	Change the content of	n/a

Option	Type	Description	Accepted Values
	reference	the html that would be the avatar for the chat bubble.	
bubble.avatar.icon	class name	Generated common library provided for icon name.	n/a

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 or chat button is disabled in configuration
});
```

Options

Option	Type	Description
openDelay	number	Duration in milliseconds to delay

Option	Type	Description
		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 and is currently not visible.	n/a
rejected	When the chat button is either not enabled in the configuration, or it's already visible.	'Chat button is not enabled in the configuration, or already visible. Ignoring command.'

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 id='cx_chat_information'>Example text</div>',
        hideFooter: false

}).done(function(e){

        // WebChat successfully shows overlay

}).fail(function(e){

        // WebChat isn't open
});
```

Options

Option	Type	Description	Accepted Values
html	string or HTML reference	The HTML content you want to display in the overlay. **Important** The `id` attribute value of the HTML content can be set to `cx_chat_information`. This supports a screen reader's ability to announce the overlay's content to the user, as recommended by WCAG.	n/a
hideFooter	boolean	Normally the overlay appears between the	true, false

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

Resolutions

Status	When	Returns
resolved	When WebChat is open and the overlay opens.	n/a
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.	n/a
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){ /* sample code */ });
```

Name	Description	Data	Introduced / Updated
ready	WebChat is initialized and ready to accept commands.	n/a	
opened	The WebChat widget has appeared on screen.	Metadata	
started	The WebChat has successfully started.	Metadata	
submitted	The user has submitted the form.	Metadata	9.0.002.06
rejected	When the chat session fails to start. Typically due to form validation or network errors.	Metadata	9.0.014.07
completed	The Chat session ended after agent is successfully connected to WebChat.	Metadata	
cancelled	The Chat session ended before agent is connected to WebChat.	Metadata	
closed	The WebChat widget has been removed from the screen.	Metadata	
minimized	The WebChat widget has been changed to a	n/a	

Name	Description	Data	Introduced / Updated
	minimized state.		
unminimized	The WebChat widget has been restored from a minimized state to the standard view.	n/a	
messageAdded	When a message is added to the transcript, this event will fire.	Returns an object containing two properties: 'data' and 'html'. 'data' contains the JSON data for the message, while 'html' contains a reference to the visible message inside the chat transcript.	

Metadata

Interaction Lifecycle

Every WebChat interaction has a sequence of events we 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
submitted
rejected
completed
closed
```

Lifecycle Scenarios

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

The user opened WebChat but changed their mind and closed it without starting a chat session:

```
ready -> opened -> cancelled -> closed
```

The user started a chat session but ended it before an agent connected. Perhaps it was taking too long to reach someone:

```
ready -> opened -> submitted -> started -> cancelled -> closed
```

The user started a chat, but the chat fails to start:

```
ready -> opened -> submitted -> rejected
```

The user started a chat, met with an agent, and the session ended normally:

```
ready -> opened -> submitted -> 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	Introduced / Updated
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.	

Name	Type	Description	Introduced / Updated
form	object	An object containing the form parameters when the form is submitted.	9.0.002.06
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.	
rejected	integer (timestamp)	Timestamp indicating when the chat session was rejected. Rejected refers to when a chat session fails to start.	9.0.014.07
completed	integer (timestamp)	Timestamp indicating when the chat session ended normally. Completed refers to when a user or agent ends a chat after an agent connected.	
closed	integer (timestamp)	Timestamp indicating when WebChat was closed.	
agentReached	integer (timestamp)	Timestamp indicating when the first agent was reached, if any.	
supervisorReached	integer (timestamp)	Timestamp indicating when the first agent supervisor was reached, if any.	
elapsed	integer (milliseconds)	Total elapsed time in milliseconds from when the user started the chat session to when the chat session ended.	
waitingForAgent	integer (milliseconds)	Total time in milleseconds waiting for an agent from when the user started the chat session to when an agent connected to the session.	

Name	Type	Description	Introduced / Updated
id	string	A Unique identifier of a chat session that helps to identify the instance of that session and its associated events.	

Customizable Chat Registration Form

Introduced: 9.0.000.08

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 an object definition that defines the layout, input type, label, and attributes for each input. You can set the default registration form definition in the `_genesys.widgets.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

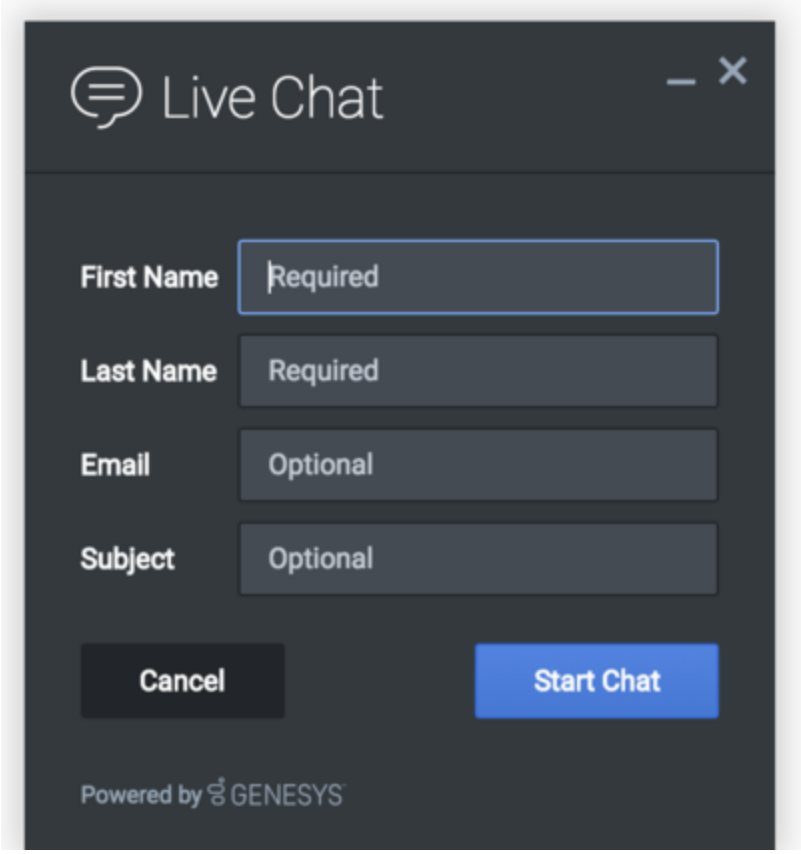
The following example is the default object used to render WebChat's registration form. This is a very simple definition that does not use many properties.

> **Important**
> You can define *any* number of inputs here, of *any* supported type, in *any* combination. Our example below simply demonstrates how WebChat defines its default form internally.

```
{
        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 definition will result in this output:



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

Property	Type	Default	Description
			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.myCusto For more information, see the Labels section.
wrapper	HTML string	“<table></table>"	Each input exists in its own row in the form. By default this is a table-row with the label in the left cell and the input in the right cell. You can redefine this wrapper and layout by specifying a new HTML row structure. See the Wrappers section for more info. The default wrapper for an input is "
validate	function		Define a validation function for the input that executes when the input loses focus (blur) or changes value. Your function must return `true` or `false`. True to indicate it passed, false to indicate it failed. If your validation fails, the form will not submit and the invalid input will be highlighted in red. See the Validation section for more details and examples.
validateWhileTyping	boolean	false	Execute validation on keypress in addition to blur and change. This ignores non-character keys like shift, ctrl, and alt.

Property	Type	Default	Description
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, {text: ‘Option 1’, value: ‘1’} for a selectable option, and {text: "Group 1", group: true} for an option group).

HTML Attributes

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

Example

```
{
        id: "cx_webchat_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:webchat.ChatFormPlaceholderFirstName",
        label: "@i18n:webchat.ChatFormFirstName"
}
```

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

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

Form Submit

Custom input field form values are submitted to the server as key value pairs under the userData section of the form submit request, where input field names will be the property keys. During the submit, this data is merged along with the userData defined in the WebChat open command.

> **Important**
>
> Depending on the API used (PureEngage V2 API or Genesys Cloud) the payload structure in the request can vary for each, but the section below explains how the form data is submitted by the WebChat UI plugin when using custom forms.

Below is the internal form data object defined in the WebChat Plugin by default. Since firstname, lastname, nickname, email, and subject are reserved keywords, users are not allowed to have custom fields with the same name.

```
{
  firstname: '',
  lastname: '',
  nickname: '',
  email: '',
  subject: '',
  userData: {}
}
```

Example

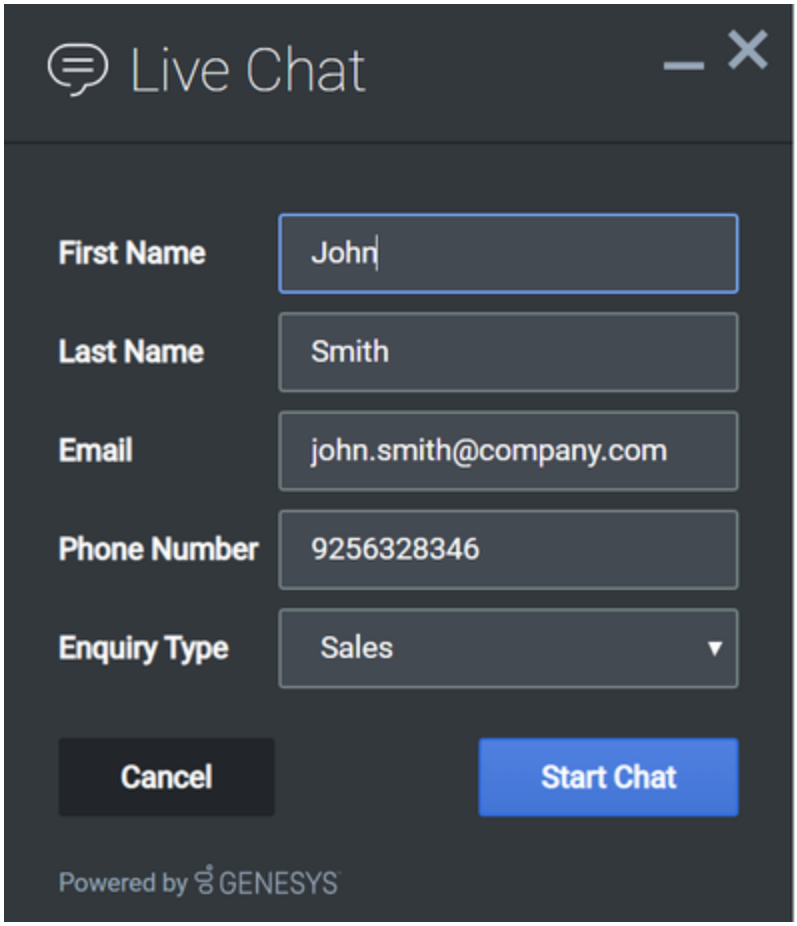
The example below shows how the custom form data given in the WebChat form fields have been mapped as a form data object.

The form fields with reserved keywords like firstname, lastname, and email will be sent as top level and the rest of the fields will be sent under userData to the WebChatService plugin.

Once the form data object is sent to the WebChatService plugin, it will parse and send in the payload request.

```
{
    "wrapper":"<table></table>",
    "inputs":[
        {
            "id":"cx_webchat_form_firstname",
            "name":"firstname",
            "type":"text",
            "maxlength":"100",
            "placeholder":"@i18n:webchat.ChatFormPlaceholderFirstName",
            "label":"@i18n:webchat.ChatFormFirstName",
            "value":"John"
        },
        {
            "id":"cx_webchat_form_lastname",
            "name":"lastname",
            "type":"text",
            "maxlength":"100",
            "placeholder":"@i18n:webchat.ChatFormPlaceholderLastName",
            "label":"@i18n:webchat.ChatFormLastName",
            "value":"Smith"
        },
        {
            "id":"cx_webchat_form_email",
            "name":"email",
```

```
            "type":"text",
            "maxlength":"100",
            "placeholder":"@i18n:webchat.ChatFormPlaceholderEmail",
            "label":"Email",
            "value":"john.smith@company.com"
        },
        {
            "id":"cx_webchat_form_phonenumber",
            "name":"phonenumber",
            "type":"text",
            "maxlength":"100",
            "placeholder":"Phone Number",
            "label":"Phone Number",
            "value":"9256328346"
        },
        {
            "id":"cx_webchat_form_enquirytype",
            "name":"enquirytype",
            "type":"select",
            "label":"Enquiry Type",
            "options":[
                {
                    "text":"Account",
                    "group":true
                },
                {
                    "text":"Sales",
                    "value":"Sales",
                    "selected":true
                },
                {
                    "text":"Credit Card",
                    "value":"credit card"
                },
                {
                    "text":"General",
                    "group":true
                },
                {
                    "text":"Warranty",
                    "value":"warranty"
                },
                {
                    "text":"Return policy",
                    "value":"returns"
                }
            ]
        }
    ]
}
```



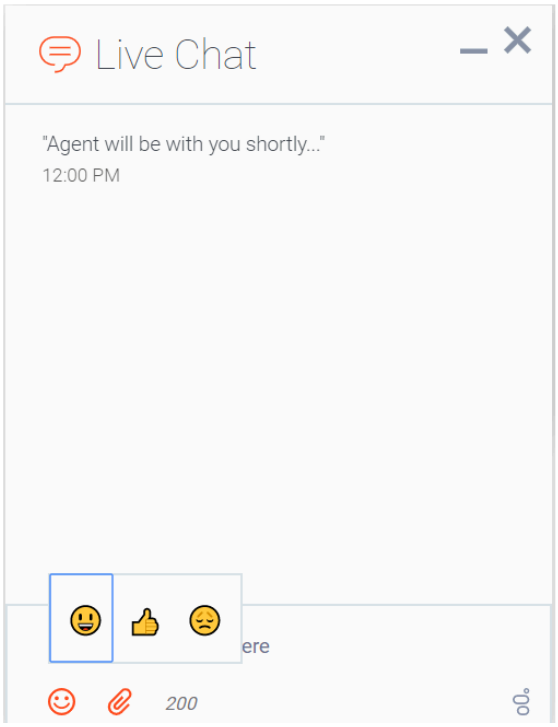

```
{
  firstname: 'John',
  lastname: 'Smith',
  email: 'john.smith@company.com',
  userData: {
    phonenumber: '9256328346',
    enquirytype: 'Sales' //value selected from the dropdown
  }
}
```

Customizable Emoji Menu

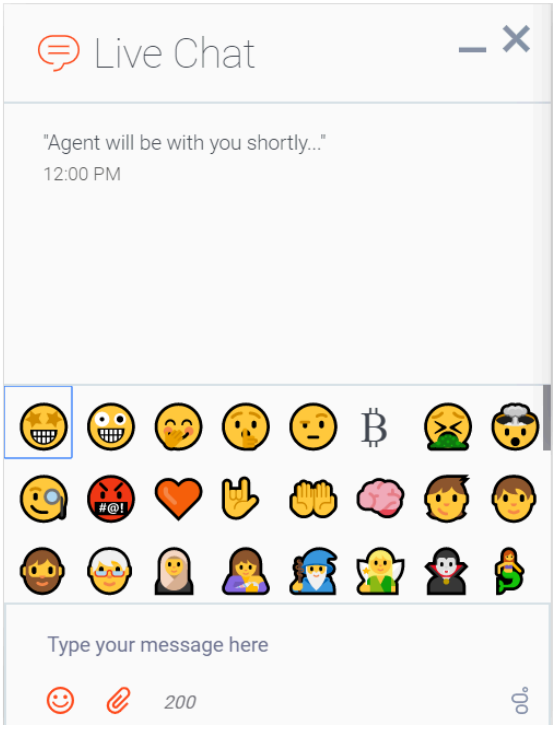
Introduction

WebChat offers a new v2 emoji menu that lets you choose emojis you want to offer.

V1 Emoji Menu



V2 Emoji Menu



Differences between v1 and v2

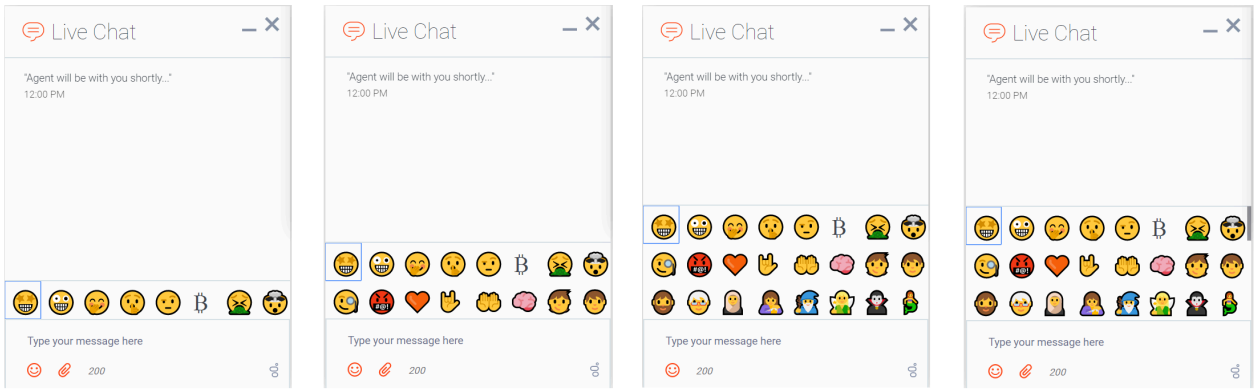
- v1 shows as a tooltip-style overlay; v2 shows as a new block between the transcript and the message input.
- v1 closes when you select an emoji or click outside the menu; v2 lets you choose multiple emojis and only closes if you click the emoji menu button again.
- v1 has three fixed emojis to choose from; v2 can show hundreds of customizable emojis in a grid layout.
- v1 menu appears in mobile mode; v2 menu is not available in mobile mode (when v2 is configured, no emoji menu button is present in mobile mode).

- v1 menu has default emojis; v2 menu does not have default emojis. It must be explicitly configured with a list of emojis.

Usage

Clicking the Emoji menu icon at the bottom-left corner of the WebChat UI will open the v2 emoji menu. The transcript will be resized to fit the emoji menu, which can vary in height depending on the number of emojis configured.

- When 1-8 emojis are configured, the menu is one row tall, and no scrollbar appears.
- When 9-16 emojis are configured, the menu is two rows tall, and no scrollbar appears.
- When 17-24 emojis are configured, the menu is three rows tall, and no scrollbar appears.
- When 25 or more emojis are configured, the menu is three rows tall and a scrollbar appears.



Configuration

The v2 emoji menu can be configured by passing a string containing emoji into the WebChat configuration or through localization.

> **Important**
>
> Note: If you define an emoji list in the WebChat configuration, it will override any emoji lists defined in localization files.

You configure the emoji list by specifying a string of emoji characters, like "😀😑😏😗". WebChat will parse this string and arrange them into the emoji menu.

```
// Configure a flat list of emoji characters
_genesys.widgets.webchat.emojiList = "😀😑😏😗😚😛😫😩😩🕹😮😶😴";
```

Emoji Display Names

You can also add names to emojis so their name will appear when you hover over them. To add names to emojis, you simply add a colon and a name, and separate each instance with a semicolon.

The format is ;😀:name;

You can only add one name to an emoji. Each emoji:name pair must be separated by a semicolon at each end to separate it from the others. A colon must be used to link the title to the emoji.

```
// Configure an emoji list with emoji names
_genesys.widgets.webchat.emojiList =
"😀:grinning;😑:expressionles;😕:confuse;😗:kissing;😙:kissing_smiling_eyes;😛:stuck_out_tongue;😟:worried;😦:frow
```

Partially Named Lists

You don't have to add names for every emoji. You can add titles to only a select few.

```
// Configure an emoji list with only a few emoji names
_genesys.widgets.webchat.emojiList =
"😀😑;😕:confuse;😗😙;😛:stuck_out_tongue;😢😦😧😮;😯:hushed;😴:sleeping;";
```

Localization

Emojis can be localized so that each language has a preferred set of emojis and emoji titles.

> **Important**
>
> Note: If you define an emoji list in the WebChat configuration, it will override any emoji lists defined in localization files.

The key name for defining an emoji list is "EmojiList".

Example:

```
{
    "en": {
        "webchat": {
            "EmojiList":
"😀:grinning;😑:expressionles;😕:confuse;😗:kissing;😙:kissing_smiling_eyes;😛:stuck_out_tongue;😟:worried;😦:frow
        }
    }
}
```

Emoji lists are defined in a localization file using the same syntax as the WebChat configuration.

SendMessageService

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: 'n3eNkgXXXXXXXXXXXX',
        dataURL: 'http://host:port/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 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

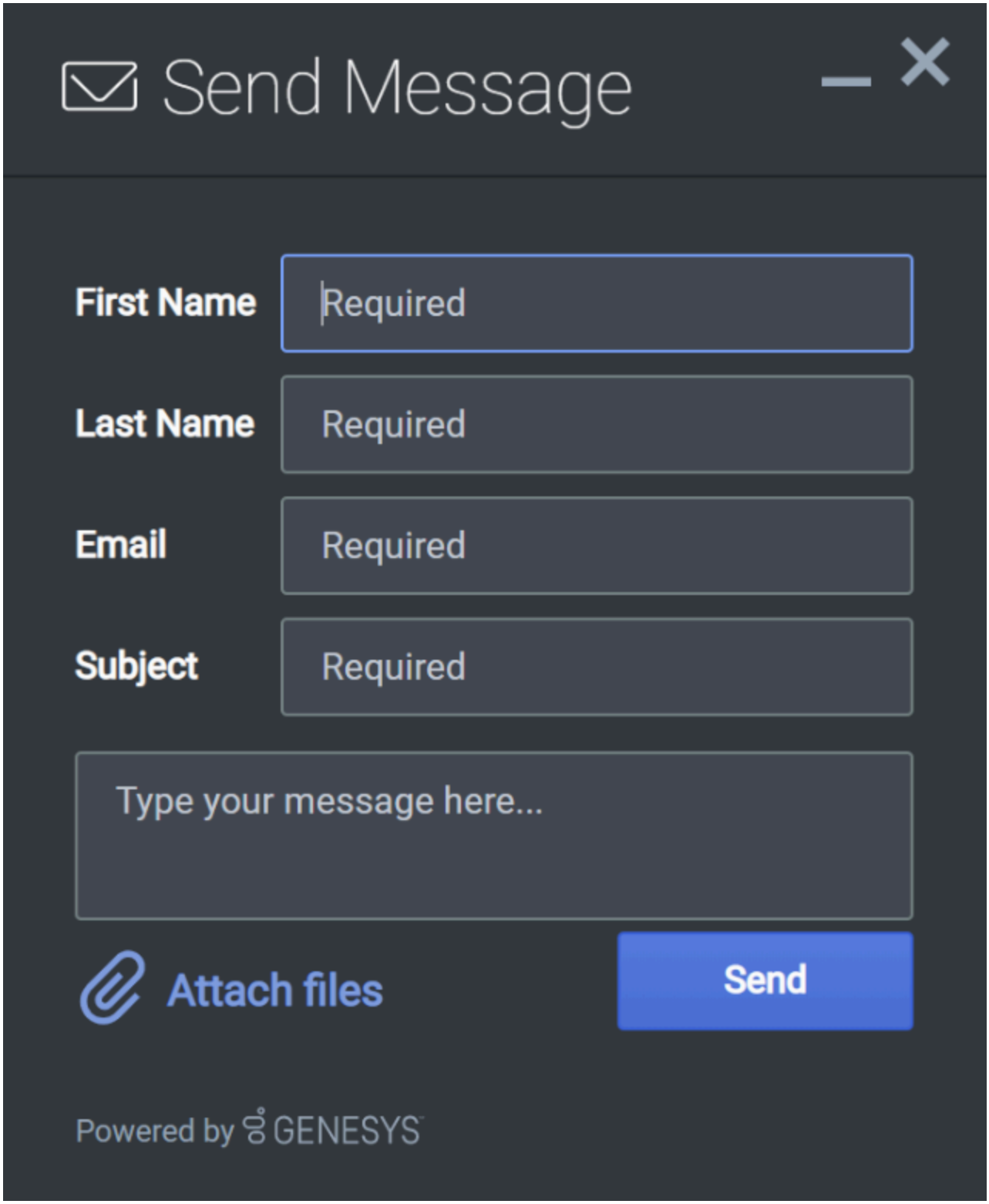
Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

> **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



- 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

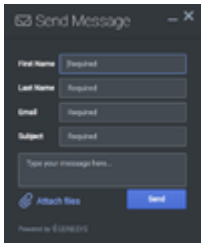
Mobile Support

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.

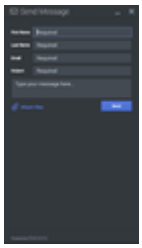
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

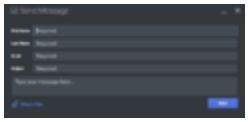
"Dark" Theme

- 

  Desktop docked view

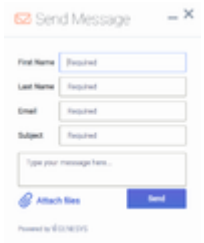
- 

  Mobile fullscreen view in portrait orientation

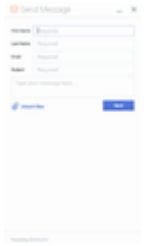
- 

Mobile fullscreen view in landscape orientation

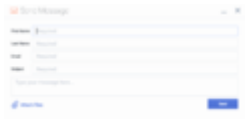
"Light" Theme

- 

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: 'n3eNkXXXXXXXXXXX',
        dataURL: 'http://host:port/genesys/2/email',

        SendMessageButton: {

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

Options

Name	Type	Description	Default	Required	Introduced / Updated
formValidation	boolean	Enable/Disable browser form validations. Note: This option is not applicable when custom forms are used.	true	n/a	
uploadsEnabled	boolean	Show/Hide the Attach Files link in the UI, will	true	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
		be shown if the value is set to true. This enables or disables the file upload feature.			
SendMessageButtonEnabled	boolean	Enable/Disable Send Message button on screen. Note: In case of running Widgets in lazy load mode, this option requires SendMessage plugin to be pre-loaded.	false	n/a	
SendMessageButtonTemplate	string	Custom HTML string template for Send Message button		n/a	
SendMessageButtonEffect	string	Type of animation effect when revealing Send Message button ('slide' or 'fade').	fade	n/a	
SendMessageButtonOpenDelay	number	Number of	1000	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
		milliseconds before displaying send message button on screen.			
SendMessageButtonEffectDuration	number	Length of animation effect in milliseconds.	300	n/a	
form	object	An object containing a custom registration form definition. The definition placed here becomes the default registration form layout for SendMessage. See Customizable SendMessage Registration Form	A basic registration form is defined internally by default	n/a	9.0.014.01

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": {
                        "SendMessageButton": "Send Message",
                        "OK": "OK",
                        "Title": "Send Message",
                        "PlaceholderFirstName": "Required",
                        "PlaceholderLastName": "Required",
                        "PlaceholderEmail": "Required",
                        "PlaceholderSubject": "Required",
                        "PlaceholderTypetexthere": "Type your message here...",
                        "FirstName": "First Name",
                        "LastName": "Last Name",
                        "Email": "Email",
                        "Subject": "Subject",
                        "Attachfiles": "Attach files",
                        "AriaAttachfiles": "Attach files link. Open a file upload dialog.",
                        "Send": "Send",
                        "AriaSend": "Send Message",
                        "Sent": "Your message has been sent...",
                        "Close": "Close",
                        "ConfirmCloseWindow": "Are you sure you want to close the Send
Message widget?",
                        "Cancel": "Cancel",
                        "AriaMinimize": "Send Message Minimize",
                        "AriaMaximize": "Send Message Maximize",
                        "AriaWindowLabel": "Send Message Window",
                        "AriaClose": "Send Message Close",
```

```
                        "AriaCloseAlert": "Alert box is closed",
                        "AriaEndConfirm": "Yes",
                        "AriaEndCancel": "Cancel",
                        "AriaOK": "OK",
                        "AriaRemoveFile": "Remove file",
                        "AriaFileIcon": "File",
                        "AriaFileSize": "File Size",
                        "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 attached data to include with 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, or it 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 and is currently not visible.	n/a
rejected	When the Send Message button is either not enabled in the configuration, or it's already visible.	'Send Message button is not enabled in the configuration, or already visible. Ignoring command.'

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, or it 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	Introduced / Updated
ready	The Send Message is initialized and ready to accept commands.	n/a	
opened	The Send Message widget has appeared on screen.	Metadata	
started	The Send Message has successfully started.	Metadata	9.0.002.06
submitted	The user has submitted the form.	Metadata	9.0.002.06
completed	The Send Message has successfully sent the message.	Metadata	9.0.002.06
cancelled	The Send Message has been closed before sending the message.	Metadata	9.0.002.06
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	

Name	Description	Data	Introduced / Updated
closed	The Send Message widget has been removed from the screen.	Metadata	

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
submitted
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 -> submitted -> 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	Introduced / Updated
proactive	boolean	Indicates SendMessage 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.	
form	object	An object containing the form parameters when the form is submitted.	9.0.002.06
opened	integer (timestamp)	Timestamp indicating when SendMessage 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 SendMessage before sending a message.	
completed	integer (timestamp)	Timestamp indicating when the message was sent successfully.	

Name	Type	Description	Introduced / Updated
closed	integer (timestamp)	Timestamp indicating when SendMessage was closed.	
elapsed	integer (milliseconds)	Total elapsed time in milliseconds from when the user started entering information to when the user cancelled or completed the interaction.	

Customizable SendMessage Registration Form

Introduced: 9.0.014.03

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

- Text
- Select
- Hidden
- Checkbox
- Textarea

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

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

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

Default Example

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

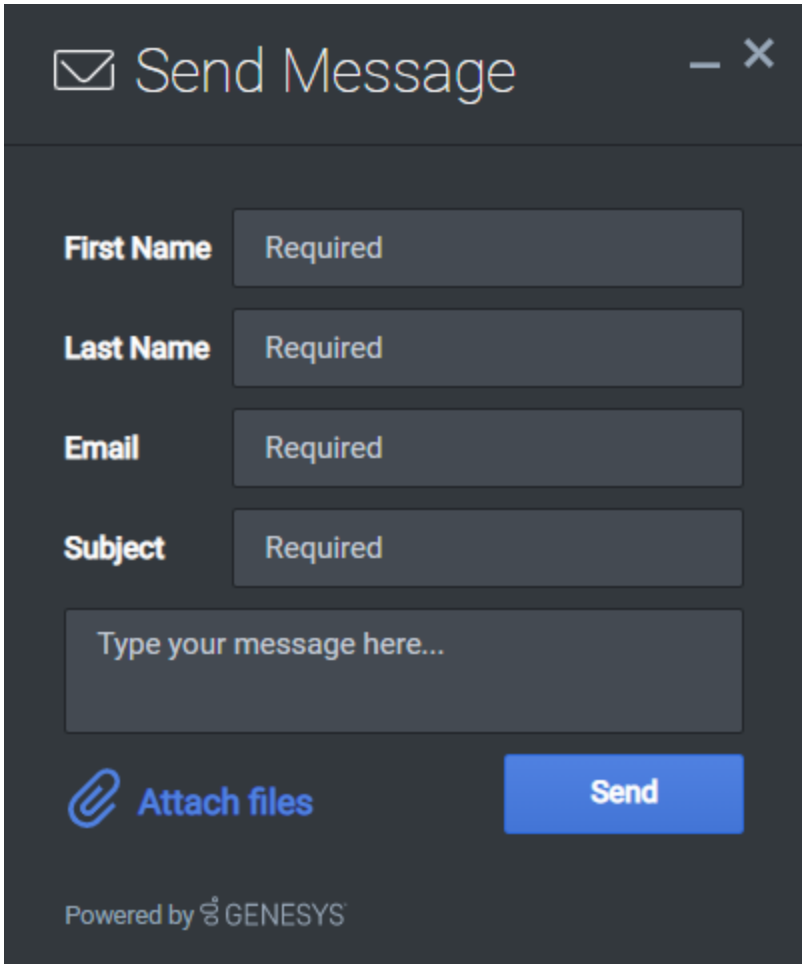
> ### Important
> You can define *any* number of inputs here, of *any* supported type, in *any* combination. Our example below simply demonstrates how SendMessage defines its default form internally.

> ### Important
> The fields with the names (firstname, lastname, email, subject, messagebody) are required for all Sendmessage custom forms. These field values are required by

> Genesys Sendmessage API to send messages.

```
{
        wrapper: "<table></table>",
        inputs: [

                {
                        id: "cx_sendmessage_form_firstname",
                        name: "firstname",
                        maxlength: "100",
                        placeholder: "@i18n:sendmessage.PlaceholderFirstName",
                        label: "@i18n:sendmessage.FirstName"
                },

                {
                        id: "cx_sendmessage_form_lastname",
                        name: "lastname",
                        maxlength: "100",
                        placeholder: "@i18n:sendmessage.PlaceholderLastName",
                        label: "@i18n:sendmessage.LastName"
                },

                {
                        id: "cx_sendmessage_form_email",
                        type: "email",
                        name: "email",
                        maxlength: "100",
                        placeholder: "@i18n:sendmessage.PlaceholderEmail",
                        label: "@i18n:sendmessage.Email"
                },

                {
                        id: "cx_sendmessage_form_subject",
                        name: "subject",
                        maxlength: "100",
                        placeholder: "@i18n:sendmessage.PlaceholderSubject",
                        label: "@i18n:sendmessage.Subject"
                },

                {
                        id: "cx_sendmessage_form_messagebody",
                        type: "textarea",
                        name: "messagebody",
                        rows: "2",
                        placeholder: "@i18n:sendmessage.PlaceholderTypetexthere",
                        label: false
                }
        ]
}
```

Using this definition will result in this output:



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

Property	Type	Default	Description
			"textarea".
label	string		Sets the text for the label. If no value is 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.myCusto For more information, see the Labels section.
wrapper	HTML string	"	
{label}	{input}		
Each input exists in its own row in the form. By default this is a table row with the label in the left cell and the input in the right cell. You can redefine this wrapper and layout by specifying a new HTML row structure. See the Wrappers section for more info. The default wrapper for an input is "			
{label}	{input}		
validate	function		Defines a validation function for the input that executes when the input loses focus (blur) or changes value. Your function must return `true` or `false`. True to indicate it passed, false to indicate it failed. If your validation fails, the form does not submit and the invalid input is highlighted in red. See

Property	Type	Default	Description
			the Validation section for more details and examples.
validateWhileTyping	boolean	false	Executes 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, {text: 'Option 1', value: '1'} for a selectable option, and {text: "Group 1", group: true} for an option group).

HTML Attributes

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

Example

```
{
        id: "cx_sendmessage_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:sendmessage.PlaceholderFirstName",
        label: "@i18n:sendmessage.FirstName"
}
```

In this example, `id`, `name`, `maxlength`, and `placeholder` are all standard HTML attributes for the text input element. Whatever values are set here are 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_sendmessage_form_firstname"
 name="firstname" maxlength="100" placeholder="Required"></input>
```

Labels

A label tag is 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 automatically links 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 SendMessage 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_sendmessage_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:sendmessage.PlaceholderFirstName",
        label: "@i18n:sendmessage.FirstName",
        wrapper: "<tr><th>{label}</th><td>{input}</td></tr>" /* input row wrapper */
}
```

The {label} identifier is optional. Omitting it allows 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_sendmessage_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:sendmessage.PlaceholderFirstName",
        label: "@i18n:sendmessage.FirstName",

        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 SendMessage 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 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.

Argument	Type	Description
Common	Function Library	Widget’s internal Common library of functions and utilities. Use if needed.

Form Submit

Custom Input field form values are submitted to the server as key value pairs under the userData section of the form submit request, where input field names will be the property keys. During the submit, this data is merged along with the userData defined in the SendMessage open command.

GWE

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 {'WebChat.opened': true}	<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

smartInvite	boolean	If set to true you will be able to see proactive invites even when webchat, sendmessage, callback, or callus is open.	false	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	Contaienr 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

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.widgets.cobrowse'. Configuration objects for the Co-browse application can be set in either 'window._genesys.cobrowse' or 'window._genesys.widgets.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

Usage

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

In the below i18n schema, 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 the localization strings below.

Localization strings

The localization strings for the Co-browse plugin are defined in the Co-browse documentation. Refer to the Co-browse documentation to find the string names and values, and define the same in the 'cobrowse' section of the Widgets i18n JSON file, as shown in the below i18n schema.

Note: The localization string names must be the same, but the values do not need to be the same and can be customized as needed.

i18n JSON Schema

```
{
        "en": {
                "cobrowse": {}
        }
}
```

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

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',
                i18n: 'i18n.json',
                mobileMode: 'auto',
                mobileModeBreakpoint: 600,
                debug: true,
                downloadGoogleFont: true,
                googleFontUrl: 'https://apps.mypurecloud.com/webfonts/roboto.css',
                header: {'Authorization': 'value'},
                cookieOptions: {
                        secure: true,
                        domain: 'genesys.com',
                        path: '/',
                        sameSite: 'Strict'
                }
        },
        onReady: function(){

                // Do something on Widgets ready
        }
}
```

Options

Name	Type	Description	Default	Required	Introduced / Updated
main.themes	object	An object list	{dark: 'cx-	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
		containing the CSS classname for each theme. The property names are used to select the theme in the 'theme' property, for example {dark:'cx-theme-dark', light:'cx-theme-light', 'red':'cx-theme-red', 'blue':'cx-theme-blue'}. Where 'dark' and 'light' 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.	theme-dark', light: 'cx-theme-light'}		
main.theme	string	Selects the theme to apply to Genesys Widgets from the 'themes' object. Uses the property name of the theme. for example using the example from 'themes' above, possible values for this could be 'dark', 'light', 'red', 'blue'.	dark	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
main.lang	string	Select the language to use from the 'i18n' language pack. Language codes are selected by the customer. Any language code format can be used as long as this property matches one of the language codes in your i18n language pack. For more information about localization, see localization.	en	n/a	
main.i18n	URL string or JSON	Either a path to a remote i18n.json language pack file or an inline JSON language pack definition. For more information about language packs, see localization.	Default English language strings are built into each widget and are displayed by default. Defining this i18n language pack overrides the built-in strings.	n/a	
main.header	object	An object containing a key value pair for the authorization header.	n/a	n/a	9.0.002.06
main.preload	array	(For use with lazy-loading only) A list of plugins you want pre-loaded at startup. You may want certain plugins, such as SideBar, to be	none	When lazy loading Widgets	

Name	Type	Description	Default	Required	Introduced / Updated
		shown on screen as soon as possible; to do so, you may add 'sidebar' to this preload plugins array so it will be loaded after Widgets starts up. The names you add to the list must match the first part of the plugin filename you wish to load. Example: 'sidebar' will load 'sidebar.min.js' from the 'plugins/' folder. All filenames are lowercase. `Note: This preload array is intended for use when running widgets in lazy-loading mode. You may also use this to pre-load your own custom-made plugins.`			
main.mobileMode	boolean/string	Mobile Mode setting. true = Force Mobile Mode on all devices. false = Disable Mobile Mode completely. 'auto' = Genesys Widgets Automatically switches between Mobile and Desktop Modes	auto	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
		using the 'mobileModeBreakpoint' property and UserAgent detection.			
main.timeFormat	number/string	This sets the time format for the timestamps. It can be 12 or 24.	12	n/a	
main.mobileModeBreakpoint	Breakpoint	The breakpoint width in pixels where Genesys Widgets will switch to Mobile Mode. Breakpoint checked at startup only.	600	n/a	
main.debug	boolean	Enable debug logging from the bus to appear in the browser console.	false	n/a	
main.customStylesheetID	string	The HTML ID of a <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.	n/a	n/a	
main.downloadGoogleFont	boolean	By default, Genesys Widgets downloads and uses the Google Font 'Roboto'. To	true	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
		disable this download, set this value to false.			
main.googleFontUrl	string	The string used to refer the URL where the Google Fonts are hosted in Genesys Hosted Repository. You can configure one of the Genesys Hosted region font URLs specified here, Genesys Web Fonts. `Note: This Option is only applicable when the downloadGoogleFont option is set to true.`	https://apps.mypurecloud.com/webfonts/roboto.css	n/a	9.0.018.00
main.deploymentId	String	The string used to customize cookie names so that multiple Widgets deployments can run in the same domain.	n/a	n/a	9.0.006.02
main.cookieOptions	object	An object containing cookie attributes that applies globally to all Widgets. The following cookie attributes are supported:	{sameSite:'Strict'}	n/a	9.0.017.01

Name	Type	Description	Default	Required	Introduced / Updated
		1. 'secure' - Either true or false, indicating if the cookie transmission requires a secure protocol (https). 2. 'domain' - A string indicating a valid domain where the cookie should be visible. 3. 'path' - A string indicating the path where the cookie is visible. 4. 'expires' - Specifies the number of days, either from time of creation or from a date instance, until the cookie is to be removed. 5. 'sameSite' - This maps to the cookie SameSite attribute allowing the cookie to be restricted to a first-party or same-site context. It can take any of the supported values that SameSite attribute			

Name	Type	Description	Default	Required	Introduced / Updated
		takes. 'domain' and 'path' can be used to make cookies compatible with environments that use a non FQDN URL, such as an intranet hostname. However, the domain should only be manually set in production if the automated values are causing problems. Otherwise, rely on the automated domain and path. Note: The values are automatically set by Widgets to support cross-sub-domain cookies. Modifying these options overrides the automated values and might break cross-sub-domain cookie support if not properly set. For usage, please refer to the above example.			
onReady	function	A callback function that is invoked when the Widgets are ready and initialized with the configuration provided.	none	n/a	

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 window._genesys.main.themes. Default themes are 'light' and 'dark'.

Resolutions

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

getTheme

Get the CSS classname for the currently selected theme.

Example

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

        // App got theme successfully
        // e == CSS classname for current theme

}).fail(function(e){

        // App failed to get theme
});
```

Resolutions

Status	When	Returns
resolved	Always	CSS classname for the currently selected theme. For example 'cx-theme-light'.
rejected	Never	n/a

reTheme

Accepts an HTML reference (either string or jQuery wrapped set) and applies the proper CSS Theme Classname to that HTML and returns it back. When widgets receive the 'theme' event from App, they pass-in their UI containers into App.reTheme to have the old theme classname stripped and 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	n/a
rejected	Never	n/a

setLanguage

Changes the language

> **Important**
> Internal use only.

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	n/a
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	n/a
rejected	Never	n/a

updateAJAXHeader

Introduced: 9.0.002.06

Updates the Authorization header.

Example

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

Resolutions

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

removeAJAXHeader

Introduced: 9.0.002.06

Removes the set Authorization header.

Example

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

Resolutions

Status	When	Returns
resolved	Always	n/a

registerExtension

Introduced: 9.0.002.06

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

Options

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

Resolutions

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

registerAutoLoad

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

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

Options

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

Resolutions

Status	When	Returns
resolved	When a plugin is added into the preload list	n/a

Status	When	Returns
rejected	Never	n/a

deregisterAutoLoad

(For use with lazy-loading only) Allows you to de-register a plugin from the preload plugins array so that it will not be pre-loaded at startup. This can be useful when there is no more active session maintained by your Widget and you don't want to show it on the screen immediately at startup.

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

Options

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

Resolutions

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

API Events

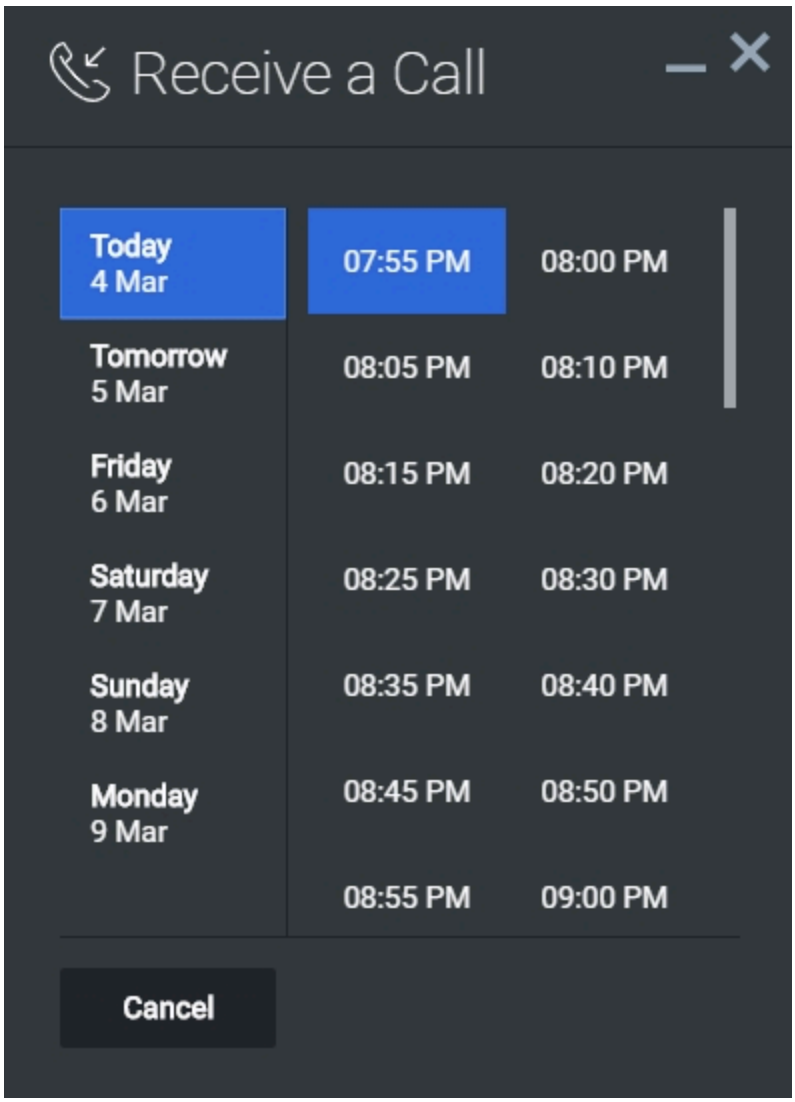
Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

> **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('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



- Configuration
- Localization
- API Commands
- API Events

Overview

Calendar widget is a UI Plugin that displays time-slots for a selected day. 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.

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

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

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,
        hideUnavailableTimeSlots: false

        calendarHours: {

                interval: 10,
                allDay: {

                        openTime: '09:00',
                        closeTime: '23:59'
                }
        }
};
```

Options

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

Name	Type	Description	Default	Required
		12 or 24.		
hideUnavailableTimeSlots	boolean	Show/hide the unavailable callback time slots.	false	n/a
calendarHours.interval	number	The time interval between each consecutive timeslot displayed on calendar.	15	n/a
calendarHours.allDay.openTime	string	Opening time in 'HH:MM' 24 Hr format.	17:00	n/a
calendarHours.allDay.closeTime	string	Closing time in 'HH:MM' 24 Hr format.	23:59	n/a

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",
```

```
                        "CalendarTitle": "Schedule a Call",
                        "CalendarOkButtonText": "Okay",
                        "CalendarError": "Unable to fetch availability details.",
                        "CalendarClose": "Cancel",
                        "AriaWindowTitle": "Calendar Window",
                        "AriaCalendarClose": "Cancel the Calendar and Go Back to the Callback
Registration",
                        "AriaYouHaveChosen": "You have chosen",
                        "AriaNoTimeSlotsFound": "No time slots found for selected date"
                }
        }
}
```

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.

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 available time slots in the Calendar plugin for the selected Date. Note that, after calling this command, the internal `showAvailability` value is set to `true` for this session and the Calendar only shows the available time slots when switching between other dates.

Resolutions

Status	When	Returns
resolved	When 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> }
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

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	callback
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

```
// If using Callback API v1

 window._genesys.widgets.callback = {

        apikey: 'n3eNkgXXXXXXXXOXXXXXXXXA',
        dataURL: 'http://host:port/genesys/1/service/callback/samples',
        userData: {},
        countryCodes: true
};

// If using Callback API v3

window._genesys.widgets.callback = {

        apikey: 'n3eNkgXXXXXXXXOXXXXXXXXA',
        apiVersion: 'v3',
        serviceName: 'service',
        dataURL: 'http://host:port/callbacks',
        userData: {},
        countryCodes: true
};
```

Options

Name	Type	Description	Default	Required	Accepted Values
apikey	string	Apigee Proxy secure token. If apiVersion is v3, this holds the x-api-key value.	n/a	Yes, if using Apigee Proxy	n/a
dataURL	URL String	URL to the API endpoint for Callback	n/a	Always	n/a
apiVersion	string	Version of	'v1'	Yes, if using	'v1', 'v3'

Name	Type	Description	Default	Required	Accepted Values
		Callback API Note: This value determines the version of Callback API in GMS/v3.		Callback v3 dataURL	
serviceName	string	Service Name of Callback API in v3	n/a	Yes, if using Callback v3 dataURL	n/a
userData	object	Arbitrary attached data to include while scheduling a callback	{}		n/a
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout	3000		n/a

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',
        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.

schedule

Schedule a callback service with the GMS callback schedule API.

Example

```
// If using Callback API v1

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

```
        userData: {}
        firstname: 'Bob',
        lastname: 'Jones',
        email: 'b.jones@mail.com',
        subject: 'product questions',
        desiredTime: '2017-03-03T00:24:17.804Z',
        phonenumber: '4151110000'

});

// If using Callback API v3

oMyPlugin.command('CallbackService.schedule', {

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

});
```

Options

Option	Type	Description
firstname	string	Receive a Call entry Form Data: 'firstname'.
lastname	string	Receive a Call entry Form Data: 'lastname'.
phonenumber	string	Receive a Call entry Form Data: 'phonenumber'.
subject	string	Receive a Call entry Form Data: 'notes'.
email	string	Receive a Call entry Form Data: 'email'.
desiredtime	string	The preferred desired time user would like to get the callback scheduled. Time should be in UTC format.
userData	object	Arbitrary data that is to be attached with callback schedule. Properties defined here will be merged with default userData set in the configuration object. If Genesys Web Engagement (GWE) is enabled, this userData also includes visitID, globalVisitID and pageID.
serviceName	string	Service Name of Callback API to

Option	Type	Description
		be passed if the apiVersion is v3.

Resolutions

Status	When	Returns
resolved	When server confirms callback is scheduled	200 OK AJAX Response - Schedule Callback `For Callback API v3, refer to 'Responses' in Schedule Callback V3`
rejected	When selected timeslot is not available	400 Bad Request AJAX Error Response - Refer to error responses under Schedule Callback `For Callback API v3, refer to 'Responses' in Schedule Callback V3`
rejected	When AJAX exception occurs	429 Too Many Requests AJAX Error Response - Refer to error responses under Schedule Callback `For Callback API v3, refer to 'Responses' in Schedule Callback V3`
rejected	When server exception occurs	500 Internal Server Error Response - Refer to error responses under Schedule Callback `For Callback API v3, refer to 'Responses' in Schedule Callback V3`
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

```
// If using Callback API v1

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
});

// If using Callback API v3

oMyPlugin.command('CallbackService.availability', {

        serviceName: 'service' // service name from callback API v3 version,
        startDate: '2017-04-03T00:24:17.804Z',
        numberOfDays: '5',
        maxTimeSlots: 20

}).done(function(e){

        // CallbackService successfully showing availability

}).fail(function(e){

        // CallbackService failed to show availability
});
```

Options

Option	Type	Description
startDate	string	The start date is specified in ISO 8601 format, using UTC as the timezone (yyyy-MM-ddTHH:mm:ss.SSSZ).
endDate	string	The end date is specified in ISO 8601 format, using UTC as timezone (yyyy-MM-ddTHH:mm:ss.SSSZ). If neither endDate nor numberOfDays is specified, the end date is assumed to be the same as the start date.
numberOfDays	string	Used as an alternative to the end date. If neither endDate nor numberOfDays is specified, the

Option	Type	Description
		end date is assumed to be the same as the start date.
maxTimeSlots	number	The maximum number of time slots to be included in the response.
serviceName	string	Service Name of Callback API to be passed if the apiVersion is v3.

Resolutions

Status	When	Returns
resolved	When server confirms the list of available callback timeslots	200 OK AJAX Response - Query Callback Availability `For Callback API v3, refer to 'Responses' in Availability Callback V3`
rejected	When timeslots are not available for selected period	400 Bad Request AJAX Response - Refer to error responses under Query Callback Availability `For Callback API v3, refer to 'Responses' in Availability Callback V3`
rejected	When AJAX exception occurs	400 Bad Request AJAX Response - Refer to error responses under Query Callback Availability `For Callback API v3, refer to 'Responses' in Availability Callback V3`
rejected	When server exception occurs	500 Internal Server Error Response - Refer to error responses under Query Callback Availability `For Callback API v3, refer to 'Responses' in Availability Callback V3`
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.

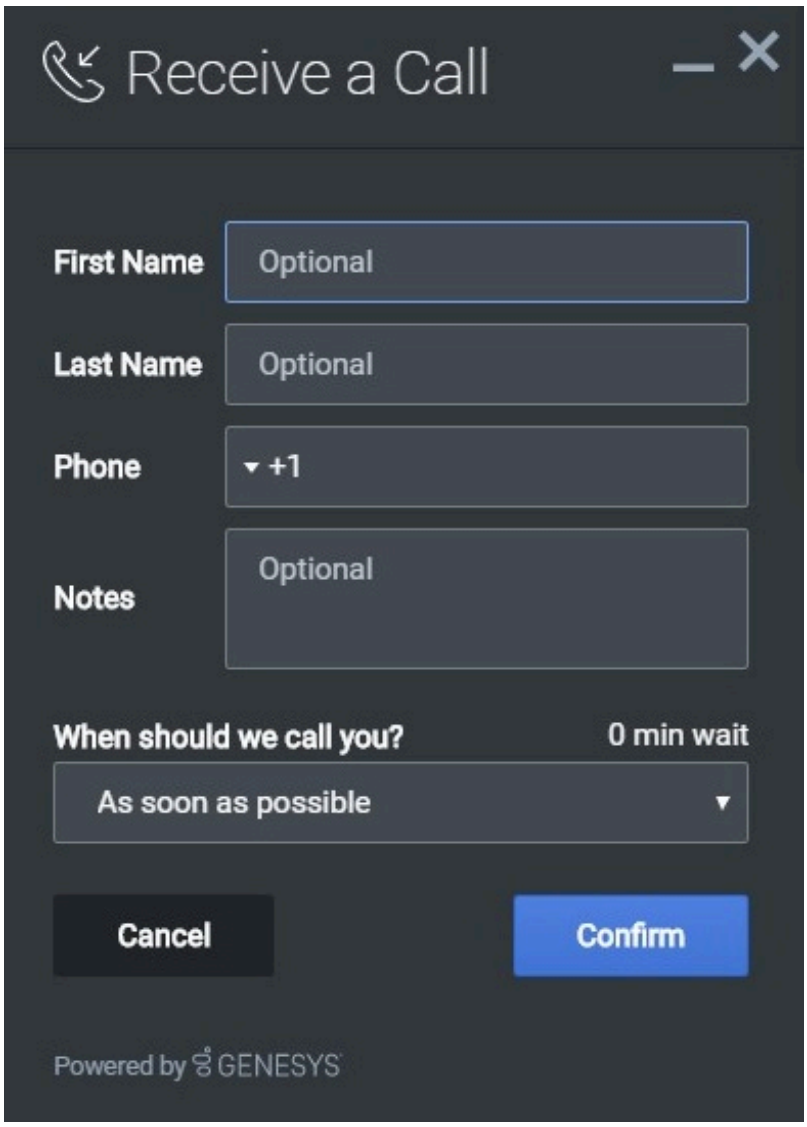
> **Important**
>
> The global bus object is a debug tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Widgets Extensions for more information about extending Genesys Widgets.

```
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.	200 OK AJAX Response - Schedule Callback. For Callback API v3, refer to 'Responses' in Schedule Callback V3
scheduleError	An error occurred between the client and the server during a callback schedule.	The JSON data returned by GMS Callback server. For more information about these error details refer to the 'Responses' status codes section under Schedule Callback API. For Callback API v3, refer to 'Responses' in Schedule Callback V3
availableSlots	Callback available slots fetched successfully.	200 OK AJAX Response - Query Callback Availability. For Callback API v3, refer to 'Responses' in Availability Callback V3
availabilityError	An error occurred between the client and the server while	The JSON data returned by GMS Callback server. For more

Name	Description	Data
	fetching the available timeslots.	information about these error details refer to the 'Responses' status codes section under Query Availability Callback API. For Callback API v3, refer to 'Responses' in Availability Callback V3

Callback



- Configuration
- Localization
- API Commands
- API Events

Overview

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

Usage

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 the 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

- 

  Choose Callback time in desktop

- 

  Mobile fullscreen view in portrait orientation

- 

  Mobile fullscreen view in landscape orientation

"Light" Theme

- 

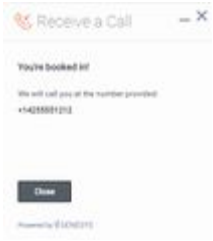
  Desktop Callback view with selected date and time

- 

  Mobile fullscreen view in portrait orientation

- 

  Mobile fullscreen view showing country codes for phone numbers

- 

  Callback confirmation in desktop

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: 'n3eNkgXXXXXXXXOXXXXXXXXA',
        dataURL: 'http://host:port/genesys/1/service/callback/samples',
        userData: {},
        countryCodes: true,
        immediateCallback: true,
        scheduledCallback: true,
        ewt: {
                display: true,
                queue: 'chat_ewt_test',
                threshold: 2000,
                immediateCallback: {
                        thresholdMin: 1000,
                        thresholdMax: 3000
                }
        }
};
```

Options

Name	Type	Description	Default	Required	Introduced / Updated
countryCodes	boolean	Enable/disable display of country codes for phone number.	true	n/a	
immediateCallback	boolean	Enable/disable the immediate (As Soon As Possible) callback option.	true	n/a	
scheduledCallback	boolean	Enable/disable the scheduling	true	n/a	

Name	Type	Description	Default	Required	Introduced / Updated
		(Pick date & time) callback option.			
form	object	An object containing a custom registration form definition. The definition placed here becomes the default registration form layout for Callback. See Customizable Callback Registration Form	A basic registration form is defined internally by default	n/a	
ewt.display	boolean	To display Estimated Wait Time (EWT) details.	true	n/a	
ewt.queue	string	EWT service channel virtual queue.	none	Always required if Estimated Waiting Time has to be displayed.	
ewt.threshold	number	If EWT is less than this threshold value (seconds), wait time will not be shown.	30	n/a	
ewt.refreshInterval	number	Wait time is updated for every time interval (seconds) defined here.	10	n/a	
ewt.immediateCallbackThresholdMin	number	If EWT is less than this minimum threshold value (seconds), then 'As Soon As Possible' option (Immediate Callback) will be disabled. This value	none	n/a	9.0.002.06

Name	Type	Description	Default	Required	Introduced / Updated
		should be configured less than or equal to above ewt.threshold value.			
ewt.immediateCallbackThresholdMax	number	If EWT is more than this maximum threshold value (seconds), then 'As Soon As Possible' option (Immediate Callback) will be disabled.	none	n/a	9.0.002.06

Localization

> **Important**
> For information on how to set up localization, please refer to the Localization Guide.

Usage

Use the 'callback' namespace when defining localization strings for the 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",
                        "CancelButtonText": "Cancel",
                        "AriaCancelButtonText": "Cancel",
                        "ConfirmButtonText": "Confirm",
                        "AriaConfirmButtonText": "Confirm",
                        "CallbackPlaceholderRequired": "Required",
                        "CallbackPlaceholderOptional": "Optional",
                        "CallbackFirstName": "First Name",
                        "CallbackLastName": "Last Name",
                        "CallbackPhoneNumber": "Phone",
                        "CallbackQuestion": "When should we call you?",
                        "CallbackDayLabels": [
                                "Sunday",
                                "Monday",
                                "Tuesday",
                                "Wednesday",
                                "Thursday",
                                "Friday",
                                "Saturday"
                        ],
                        "CallbackMonthLabels": [
                                "Jan",
                                "Feb",
                                "Mar",
                                "Apr",
```

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

API Commands

Once you've registered your 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',
                subject: 'Customer Satisfaction',
                desiredTime: 'now',
                phonenumber: '8881110000'
        },
        formJSON: {...}

}).done(function(e){

        // Callback opened successfully

}).fail(function(e){

        // Callback failed to open
});
```

Options

Option	Type	Description
form	object	Object containing form data to prefill in the callback form and optionally auto-submit the form.
form.autoSubmit	boolean	Automatically submit the callback form.
form.firstname	string	Value for the first name entry field.
form.lastname	string	Value for the last name entry field.
form.subject	string	Value for the notes entry field.
form.desiredTime	string	This value is shared by the immediate or scheduled callback drop down option in the form (in other words, As Soon As Possible or Pick date & time). A string value 'now' pre-selects the 'As Soon As Possible' option. A string value with Date Time or Date Object, is passed into this drop down option and pre-selected. During form submission, it is converted into UTC string format and sent to the server as the desired callback time.
form.phonenumber	string	Value for the phone entry field. Should be a valid telephone number, when used with a prefix '+' auto selects the country flag near the phone input field.
formJSON	object	An object containing a custom registration form definition. See Customizable Callback Registration Form
userData	object	Arbitrary data that is to be attached with callback schedule. Properties defined here will be merged with default userData set in the configuration object.

Resolutions

Status	When	Returns
resolved	When callback form is successfully opened.	n/a
rejected	When callback form is already	'already opened'

Status	When	Returns
	open.	

close

Closes the Callback UI.

Example

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

Resolutions

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

minimize

Minimize or unminimize Callback UI.

Example

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

Options

Option	Type	Description
minimized	boolean	Rather than toggling the current minimized state you can specify the minimized state directly: true

Option	Type	Description
		= minimized, false = unminimized.

Resolutions

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

showOverlay

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

Example

```
oMyPlugin.command('Callback.showOverlay', {

        html: '<div>Example text</div>'

});
```

Options

Option	Type	Description
html	string or HTML reference	The HTML content you want to display in the overlay.
hideFooter	boolean	Normally the overlay appears between the titlebar and footer bar. Set this to true to have the overlay overlap the footer to gain a bit more vertical space. This should only be used in special cases. For general use, don't set this value.

Resolutions

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

hideOverlay

Hides the slide-down overlay.

Example

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

Resolutions

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

configure

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

API Events

Once you've registered your 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	Introduced / Updated
opened	The Callback widget has appeared on screen.	metadata	
ready	The Callback widget is initialized and ready to accept commands.	n/a	
started	When the user has started filling out the Callback widget form or auto pre-filled it.	metadata	
submitted	When the user has submitted the form.	metadata	9.0.002.06
completed	When the Callback widget form is submitted successfully.	metadata	
cancelled	When the user has abandoned the interaction by closing the Callback widget before scheduling a callback.	metadata	
closed	The Callback widget has been removed from the screen.	metadata	

Metadata

Interaction Lifecycle

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

The following events are part of the Interaction Lifecycle:

```
ready
opened
started
submitted
cancelled
completed
closed
```

Lifecycle Scenarios

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

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

```
ready -> opened -> cancelled -> closed
```

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

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

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

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

> **Tip**
> For a list of all Callback events, see API Events.

Metadata

Each event in the Interaction Lifecycle includes the following block of metadata. By default, all values

are set to `false`. As the user progresses through the lifecycle of a Callback interaction, these values will be updated.

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

Reference

Name	Type	Description	Introduced / Updated
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.	
form	object	An object containing the form parameters when the form is submitted.	9.0.002.06
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.	

Name	Type	Description	Introduced / Updated
elapsed	integer (milliseconds)	Total elapsed time in milliseconds from when the user started entering information to when the user cancelled or completed the interaction.	

Customizable Callback Registration Form

Introduced: 9.0.001.04

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

- Text
- Select
- Hidden
- Checkbox
- Textarea

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

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

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

Default Example

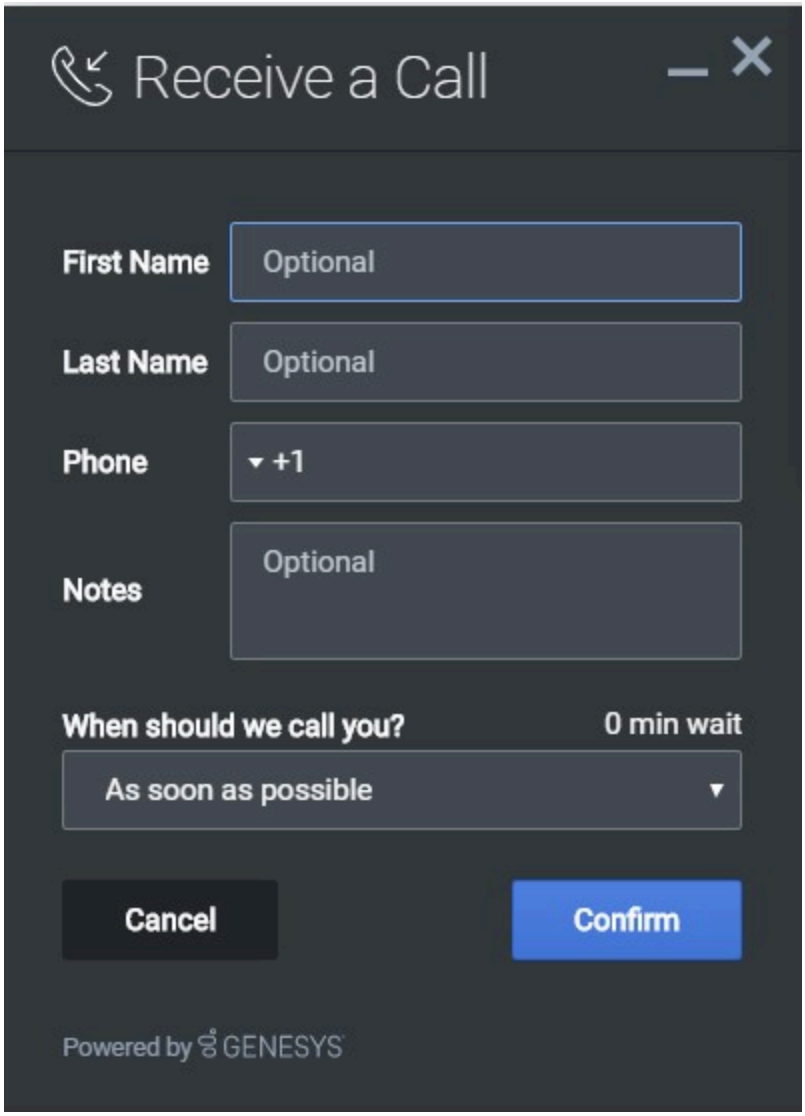
The following example is the default object used to render Callback's registration form. This is a very simple definition that does not use many properties.

> ### Important
> You can define *any* number of inputs here, of *any* supported type, in *any* combination. Our example below simply demonstrates how WebChat defines its default form internally.

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

```
{
        wrapper: "<table></table>",
        inputs: [

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

                    {
                        id: "cx_form_callback_lastname",
                        name: "lastname",
                        maxlength: "100",
                        placeholder: "@i18n:callback.CallbackPlaceholderOptional",
                        label: "@i18n:callback.CallbackLastName"
                    },

                    {
                        id: "cx_form_callback_phone_number",
                        name: "phonenumber",
                        maxlength: "14",
                        placeholder: "@i18n:callback.CallbackPlaceholderRequired",
                        label: "@i18n:callback.CallbackPhoneNumber",

                        onkeypress: function(event) {

                                // To allow only number inputs
                                return (event.charCode >= 48 && event.charCode <= 57) ||
(event.charCode == 43)
                        }
                    },

                    {
                        id: "cx_form_callback_subject",
                        name: "subject",
                        type: "textarea",
                        maxlength: "100",
                        placeholder: "@i18n:callback.CallbackPlaceholderOptional",
                        label: "@i18n:callback.CallbackNotes"
                    }
        ]
}
```

Using this definition will result in this output:



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

Properties

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

element.

Special Properties

Property	Type	Default	Description
type	string	"text"	Sets the type of input to render. Possible values are currently "text", "hidden", "select", "checkbox", and "textarea".
label	string		Set the text for the label. If no value provided, no label will be shown. You may use localization query strings to enable custom localization (for example, label: "@i18n:namespace.StringName"). Localization query strings allow you to use strings from any widget namespace or to create your own namespace in the localization file (i18n.json) and use strings from there (for example, label: "@i18n:myCustomNamespace.myCusto For more information, see the Labels section.
wrapper	HTML string	“<table></table>"	Each input exists in its own row in the form. By default this is a table-row with the label in the left cell and the input in the right cell. You can redefine this wrapper and layout by specifying a new HTML row structure. See the Wrappers section for more info. The default wrapper for an input is "
validate	function		Define a validation function for the input that executes when the input loses focus (blur) or changes value. Your

Property	Type	Default	Description
			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_callback_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:callback.CallbackPlaceholderOptional",
        label: "@i18n:callback.CallbackFirstName"
}
```

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

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

HTML Output

```
<input type="text" id="cx_callback_form_firstname" name="firstname" maxlength="100"
```

```
placeholder="Optional"></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 "<table></table>". This is the default wrapper for the Callback 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_callback_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:callback.CallbackPlaceholderOptional",
        label: "@i18n:callback.CallbackFirstName"
        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_callback_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:callback.CallbackPlaceholderOptional",
        label: "@i18n:callback.CallbackFirstName"

        validateWhileTyping: true, // default is false

        validate: function(event, form, input, label, $, CXBus, Common){

                if(input && input.val()) {    // to validate some input exits in the
firstname input field (required field)

                    return true;              // validation passed

                }else{

                    return false;             // no input exists, validation failed
                }
        }
}
```

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

Validation Function Arguments

Argument	Type	Description
event	JavaScript event object	The input event reference object related to the form input field. This event data can be helpful to perform actions like active

Argument	Type	Description
		validation on an input field while the user is typing.
form	HTML reference	A jquery reference to the form wrapper element.
input	HTML reference	A jquery reference to the input element being validated.
label	HTML reference	A jquery reference to the label for the input being validated.
$	jquery instance	Widget's internal jquery instance. Use this to help you write your validation logic, if needed.
CXBus	CXBus instance	Widget's internal CXBus reference. Use this to call commands on the bus, if needed.
Common	Function Library	Widget's internal Common library of functions and utilities. Use if needed.

Form Submit

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

Form Prefill

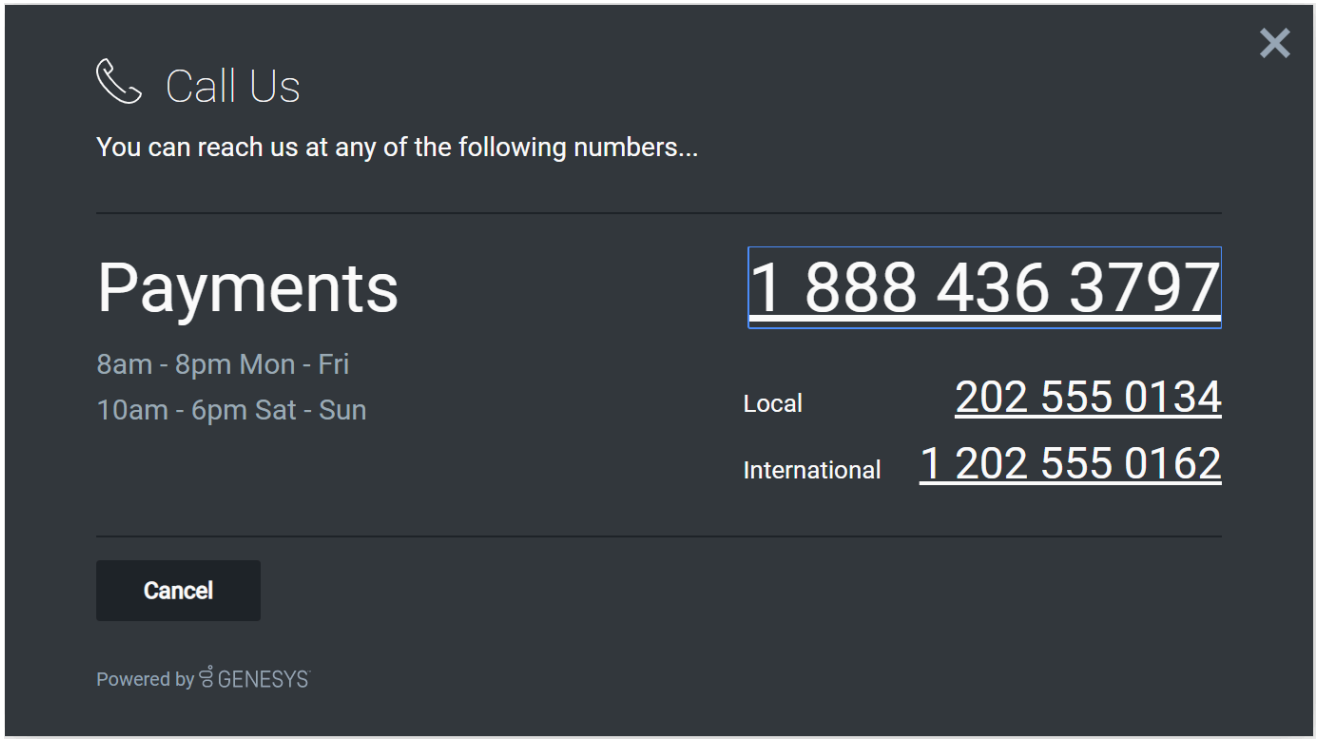
You can prefill the custom form using Callback.open command by passing the form (form data) and formJSON (custom registration form), provided the form input names in the formJSON must match with the property names in the form data.

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

```
_genesys.widgets.bus.command("Callback.open", {

        formJSON: {
                   wrapper: "<table>",
                    inputs: [{
                            id: "cx_form_phone_number",
                            name: "phonenumber",
                            maxlength: "12",
                            placeholder: "@i18n:callback.CallbackPlaceholderPhoneNumber",
                            label: "@i18n:callback.CallbackPhoneNumber"
                        }]
        },

        form: {
```

```
                phonenumber: 9453222222
        }
});
```

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.

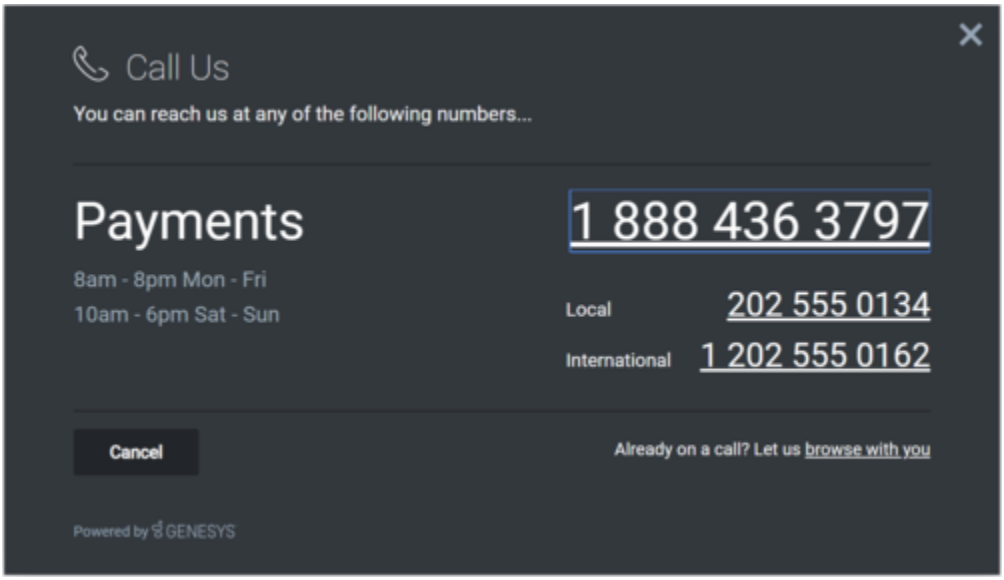
Usage

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

		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'.
		Ex:
		{
		"displayName": "Payments",
		"i18n": "Number001",
		"number": "1 202 555 0162"
		}		

hours	array	Array of strings to show stacked in the business hours section. Strings here are freeform. See screenshots for ideas.	[]	

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 role='link' tabindex='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.",
                        "AriaWindowLabel": "Call Us Window",
                        "AriaCallUsClose": "Call Us Close",
                        "AriaBusinessHours": "Business Hours",
                        "AriaCallUsPhoneApp": "Opens the phone application",
                        "AriaCobrowseLink": "Opens the Co-browse Session",
                        "AriaCancelButtonText": "Call Us Cancel"
                }
        }
}
```

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

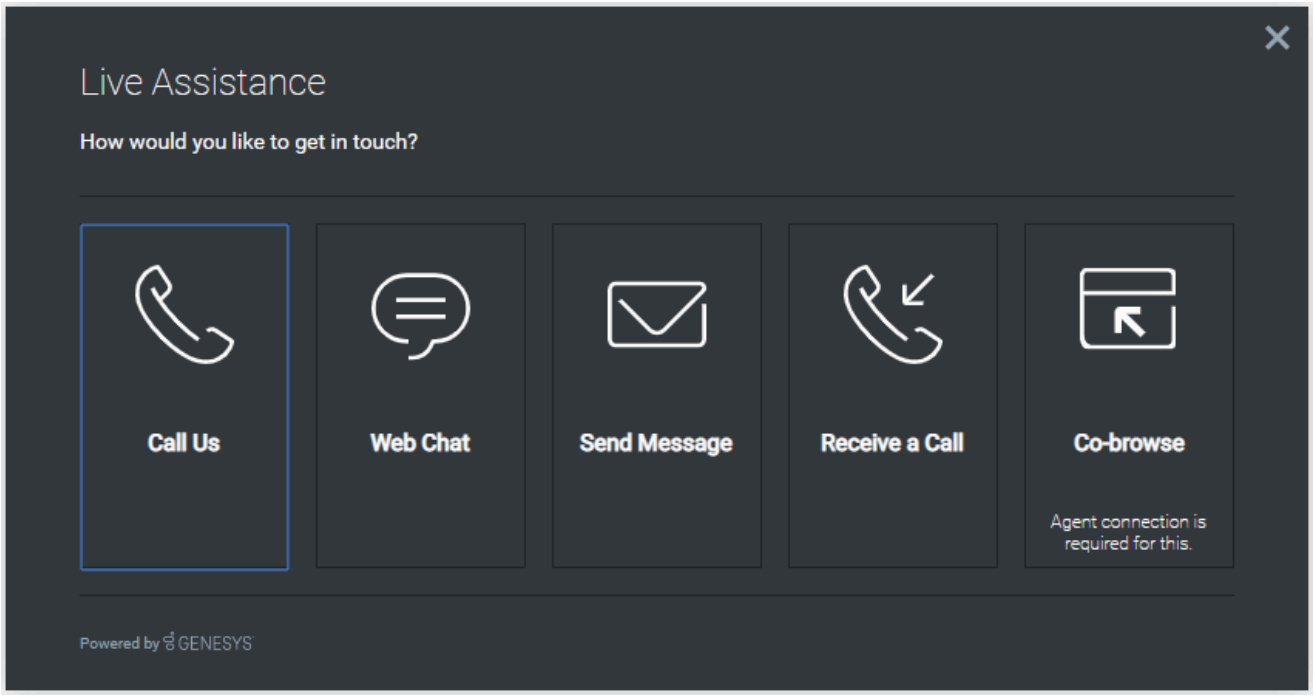
Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

> **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 additional 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 minimum

- 

Desktop overlay view showing Co-browse channel

- 

Mobile full-screen view in landscape orientation showing EWT available

- 

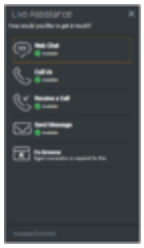
Mobile full-screen view in landscape orientation showing EWT maximum & unavailable

- 

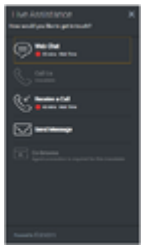
Mobile full-screen view in landscape orientation showing EWT maximum

- 

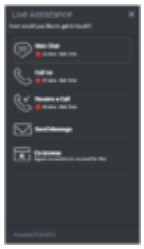
Mobile full-screen view in landscape orientation showing EWT minimum

- 

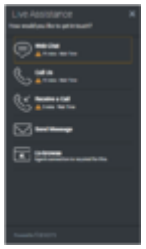
  Mobile full-screen view in portrait orientation showing EWT available

- 

  Mobile full-screen view in portrait showing EWT maximum & unavailable

- 

  Mobile full-screen view in portrait orientation showing EWT maximum

- 

  Mobile full-screen view in portrait orientation showing EWT minimum

"Light" Theme

- 

Desktop overlay view showing EWT available

- 

Desktop overlay view showing EWT maximum & unavailable

- 

Desktop overlay view showing EWT minimum

- 

Desktop overlay view showing Co-browse channel

- 

Mobile full-screen view in landscape orientation showing EWT available

- 

Mobile full-screen view in landscape orientation showing EWT maximum & unavailable

- 

Mobile full-screen view in landscape orientation showing EWT maximum

- 

Mobile full-screen view in landscape orientation showing EWT minimum

- 

Mobile full-screen view in portrait orientation showing EWT available

- 

  Mobile full-screen view in portrait showing EWT maximum & unavailable

- 

  Mobile full-screen view in portrait orientation showing EWT maximum

- 

  Mobile full-screen view in portrait orientation 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 Wait Time (EWT) details. All the channels are displayed based on the array of objects order defined in channels config. To hide a particular channel, simply remove the corresponding array object.

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

Example

```
window._genesys.widgets.channelselector = {

        ewtRefreshInterval: 10,

        channels: [{

                enable: true,
                clickCommand: 'CallUs.open',
                displayName: 'Call Us',
                i18n: 'CallusTitle',
                icon: 'call-outgoing',
                html: '<img src='http://placehold.it/100x100'>',
                ewt: {
                        display: true,
                        queue: 'callus_ewt_test_eservices',
                        availabilityThresholdMin: 300,
                        availabilityThresholdMax: 480,
                        hideChannelWhenThresholdMax: false
                        }
                },

                {
                enable: true,
                clickCommand: 'WebChat.open',
                displayName: 'Web Chat',
                i18n: 'ChatTitle',
                icon: 'chat',
                html: '',
                ewt: {
```

```
                        display: true,
                        queue: 'chat_ewt_test_eservices',
                        availabilityThresholdMin: 300,
                        availabilityThresholdMax: 480,
                        hideChannelWhenThresholdMax: false
                        }
                },

                {
                enable: true,
                clickCommand: 'SendMessage.open',
                displayName: 'Send Message',
                i18n: 'EmailTitle',
                icon: 'email',
                html: ''
                },

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

                {
                enable: true,
                name: 'CoBrowse',
                clickCommand: 'CoBrowse.open',
                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 for opening a particular widget	none	Always

Name	Type	Description	Default	Required
		when this channel is clicked on.		
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
channels[].ewt.display	boolean	To display EWT details.	true	n/a
channels[].ewt.queue	string	EWT service channel virtual queue.	none	Always
channels[].ewt.availabilityThreshold	number (seconds)	If EWT is greater than 0 min and less than this minimum threshold value (in minutes), then the EWT is shown with a yellow warning icon. `Note: Comparison is made after converting the threshold value in seconds to minutes.`	300	n/a
channels[].ewt.availabilityThreshold	number (seconds)	If EWT is greater	480	n/a

Name	Type	Description	Default	Required
		than this minimum threshold value (in minutes) and less than the maximum threshold value (in minutes), then the EWT is shown with a red alert icon. Note: Comparison is made after converting the threshold value in seconds to minutes.		
channels[].ewt.hideChannelWhenThresholdGreater	boolean	Hides this channel when EWT is greater than the 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",
                        "AriaAvailableTitle": "Available",
                        "UnavailableTitle": "Unavailable",
                        "CobrowseButtonText": "CobrowseSubTitle",
                        "CallbackTitle": "Receive a Call",
                        "CobrowseSubTitle": "Agent connection is required for this.",
                        "AriaClose": "Live Assistance Close",
                        "AriaWarning": "Warning",
                        "AriaAlert": "Alert",
                        "minute": "min",
                        "minutes": "mins",
                        "AriaWindowLabel": "Live Assistance Window"
                }
        }
}
```

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

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('ChannelSelector.configure', {

        channels: [
                {
                        enabled: true,
                        clickCommand: 'CallUs.open',
                        displayName: 'Call Us',
                        i18n: 'CallusTitle',
                        icon: 'call-outgoing',
                        html:  '',
                        ewt:{

                                display: true,
                                queue:'chat_ewt_test_eservices',
                                availabilityThresholdMin:60,
```

```
                                availabilityThresholdMax:600
                        }
                }
        ]

}).done(function(e){

        // ChannelSelector configured successfully

}).fail(function(e){

        // ChannelSelector failed to configure
});
```

Options

Option	Type	Description
ewtRefreshInterval	number	EWT is updated for every time interval (seconds) is defined.
channels	array	Array containing each channel configuration object. The order of channels is displayed based on the order defined here.
channels[].enable	boolean	Enable/disable chat channel.
channels[].clickCommand	string	The CXBus command name for opening a particular Widget when clicked on this channel.
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 EWT details.
channels[].ewt.queue	string	EWT service channel virtual queue name.
channels[].ewt.availabilityThresholdMin	Number (seconds)	If the EWT is greater than 0 minutes and less than the minimum threshold value (in minutes), then the EWT is shown with a yellow warning icon.

Option	Type	Description
		Note: Comparison is made after converting the threshold value in seconds to minutes.
channels[].ewt.availabilityThresholdMax	Number (seconds)	If the EWT is greater than the minimum threshold value (in minutes) and less than the maximum threshold value (in minutes), then the EWT is shown with a red alert icon. Note: Comparison is made after converting the threshold value in seconds to minutes.
channels[].ewt.hideChannelWhenThresholdMax	Number	Hides this channel when the EWT is greater than the 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 Wait Time (EWT) 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 EWT is displayed successfully	n/a
rejected	When StatsService fails to retrieve EWT data	'Unable to display EWT Stats in 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 EWT. 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 EWT 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 EWT details are already displayed	'Stats already enabled'
rejected	When ChannelSelector Widget is not opened	'ChannelSelector not opened to enable stats details'

API Events

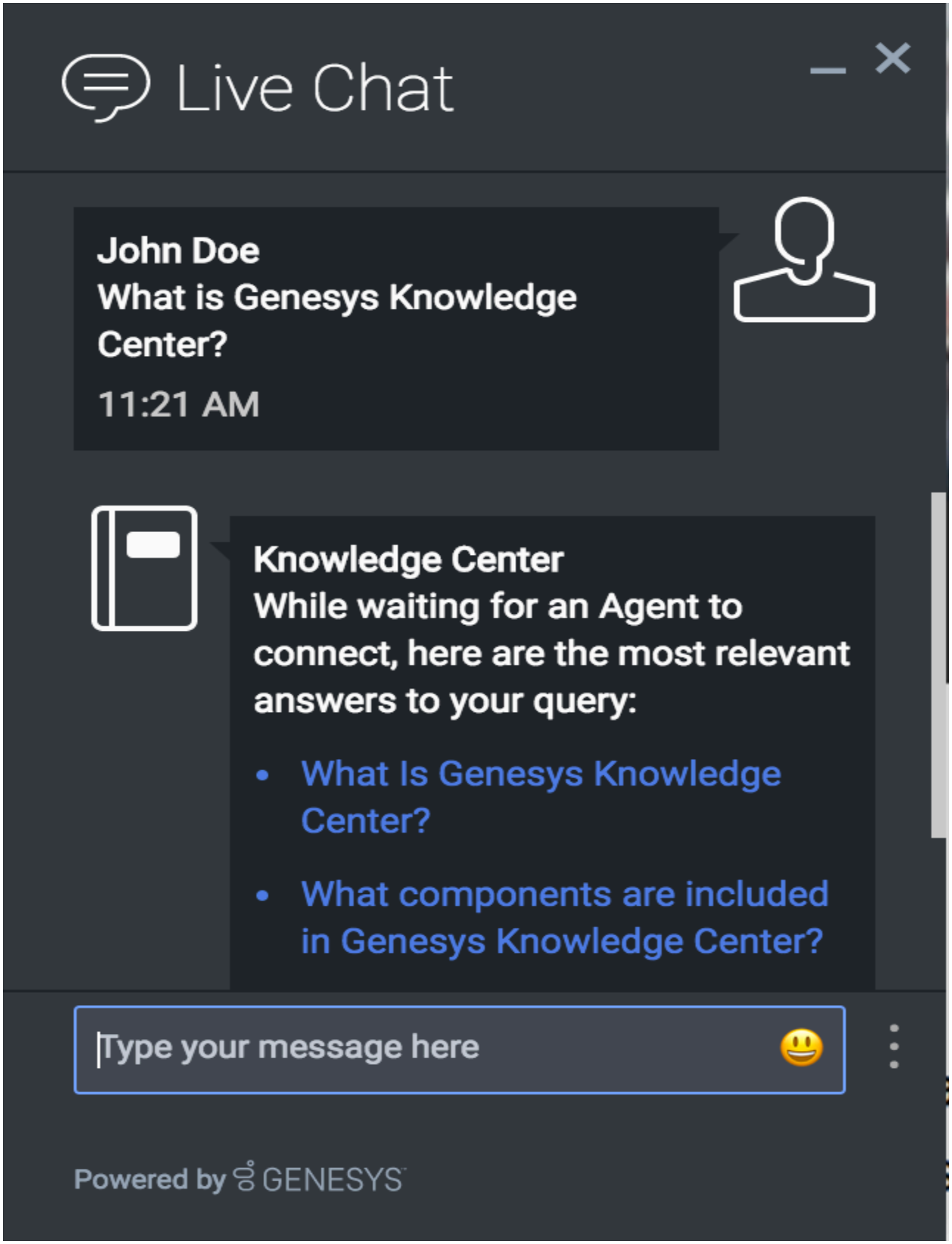
Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

> **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?
Type your message here
Powered by GENESYS

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 joines 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

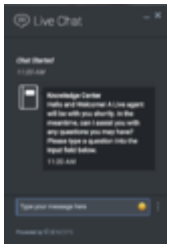
Mobile Support

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.

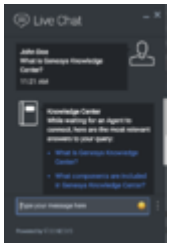
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

"Dark" Theme

- 

  Mobile fullscreen view in portrait orientation showing deflection invitation message

- 

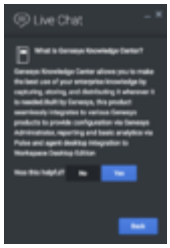
  Desktop docked view showing deflection response

- 

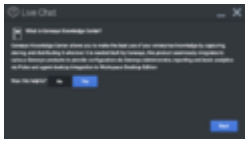
  Mobile fullscreen view in Landscape orientation showing deflection response

- 

  Mobile fullscreen view in portrait orientation showing deflection response

- 

  Desktop docked view showing document details (since 8.5.004.19)

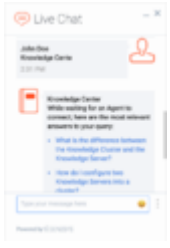
- 

  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

- 

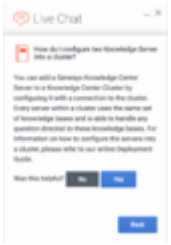
  Desktop docked view showing deflection response

- 

  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.widgets.knowledgecenter' configuration namespace and has connection and chat-deflection options.

Example

```
window._genesys.widgets.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.sessionKey	string	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.languageKey	string	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.questionKey	string	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 confugure
});
```

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 disable. 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

ClickToCallService

Overview

ClickToCallService exposes a high-level API for utilizing Genesys callback services. You can use these services for requesting a customer service number and access code for your user to dial.

Usage

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

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

Namespace

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

Type	Namespace
Configuration	clicktocall
CXBus - API Commands & API Events	ClickToCallService

Customization

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

Configuration

Description

ClickToCall and ClickToCallService share the configuration namespace '_genesys.widgets.clicktocall'. ClickToCall has UI options while ClickToCallService has connection options.

Example

```
window._genesys.widgets.clicktocall = {

        'ajaxTimeout' : 3000,
        'provideAccessCode' : true,
        'dataURL' : 'http://www.myphoneservice.org',
        'apikey' : 'YOUR_API_KEY',
        'userData' : {}
};
```

Options

Name	Type	Description	Default	Required
ajaxTimeout	Number	Sets the default ajax timeout in milliseconds.	3000	false
provideAccessCode	boolean	Enables or disables the use of a dial in access code for user verification.	true	false
dataURL	string	URL of GMS ClickToCall API endpoint.	n/a	true
apikey	string or number	Apigee Proxy secure token.	n/a	Yes, if using Apigee Proxy.
userData	object	Arbitrary JSON attached data to include while requesting ClickToCall phone number.	{}	n/a

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('ClickToCallService.requestNumber', {

        userData: {
                firstname: 'Bob',
                lastname: 'Jones'
        },
        phonenumber: '415XXXXXXX'

});
```

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.

restore

Tries to return saved session data to the UI plugin to restore the widget to its previous state.

Example

```
oMyPlugin.command('ClickToCallService.restore');
```

Resolutions

Status	When	Returns
resolved	When ClickToCallService attempts to restore data.	Restored data or empty object.
rejected	Never	n/a

requestNumber

Requests a phone number, access code and expiration time through the GMS Callback Service API (Voice - User Originated).

Example

```
oMyPlugin.command('ClickToCallService.requestNumber', {

        userData: {
                firstname: 'Bob',
                lastname: 'Jones'
        },
        phonenumber: '415XXXXXXX'

});
```

Options

Option	Type	Description
phonenumber	string	ClickToCall Entry Form Data: 'phonenumber'.
userData	object	Arbitrary data that is attached with ClickToCall phone number request. Properties defined here are 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	Always	(AJAX response data)
rejected	When AJAX exception occurs	(AJAX Response Object)

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.

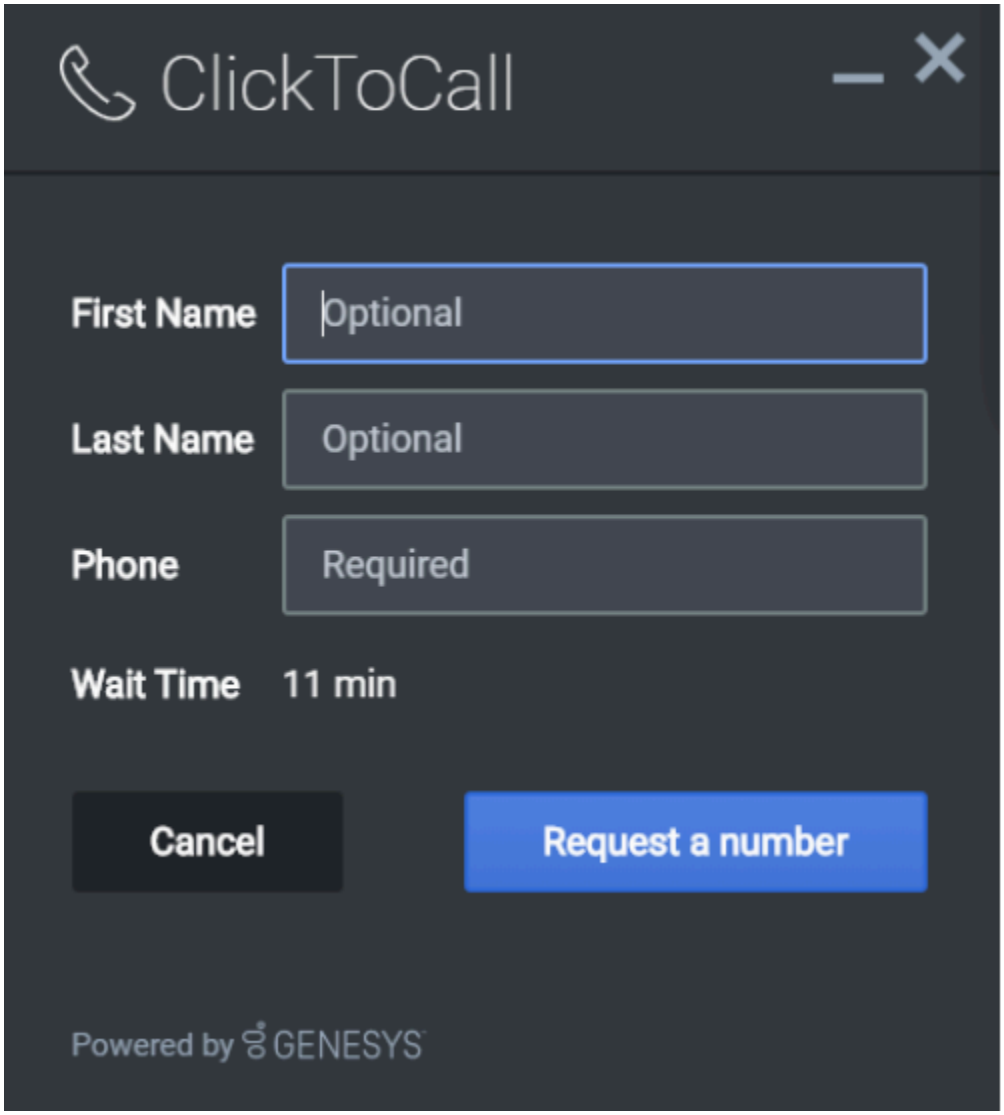
> **Important**
>
> The global bus object is a debug tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Widgets Extensions for more information about extending Genesys Widgets.

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

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

Name	Description	Data
ready	ClickToCallService is initialized and ready to accept commands.	n/a
restored	ClickToCallService has restored the currently active phone number.	{bRestoreSuccess: (boolean), sPhoneNumber: (string), sPhoneTelHref: (string), sAccessCode: (string), iExipreTime: (number)}
numberReceived	ClickToCallService has received a phone number from the server.	{sPhoneNumber: (string), sPhoneTelHref: (string), sAccessCode: (string), iExipreTime: (number)}

ClickToCall

Introduced: 9.0.001.04



- Configuration
- Localization
- API Commands
- API Events
- API Metadata
- Customizable Registration Form

Overview

The ClickToCall Widget allows customers to request a phone number to dial-in to customer service, and supports initiating the call by a single click on the button.

Usage

The ClickToCall widget can be launched manually by the following methods:

- Calling the command "ClickToCall.open".
- Configuring Sidebar to show ClickToCall in it.

Customization

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

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

Namespace

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

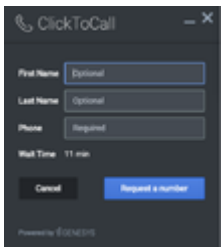
Type	Namespace
Configuration	clicktocall
i18n - Localization	clicktocall
CXBus - API Commands & API Events	ClickToCall
CSS	.cx-clicktocall

Mobile Support

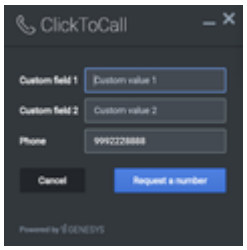
ClickToCall has full mobile support in both landscape and portrait modes, as well as being able to click the displayed phone number to dial it.

Screenshots

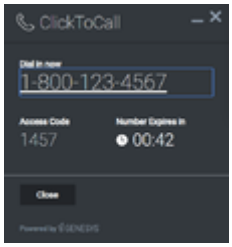
"Dark" Theme

- 

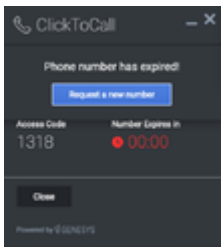
  Desktop docked view showing the default form

- 

  Desktop view showing the custom form with phone number pre-filled

- 

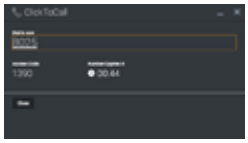
  Desktop view showing the phone number, access code and expiration timer

- 

  Desktop view showing the phone number expiration message

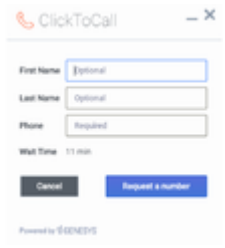
- 

  Mobile portrait view showing the default form

- 

  Mobile landscape view showing the phone number, access code and expiration timer

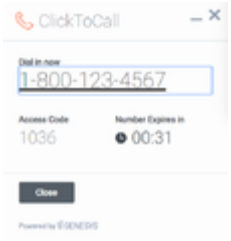
"Light" Theme

- 

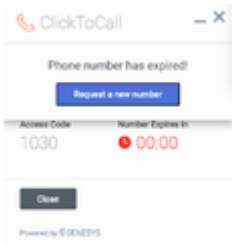
  Desktop docked view showing the default form

- 

  Desktop view showing the custom form with phone number pre-filled

- 

  Desktop view showing the phone number, access code and expiration timer

- 

  Desktop view showing the phone number expiration message

- 

  Mobile portrait view showing the default form

- 

  Mobile landscape view showing the phone number, access code and expiration timer

Configuration

Description

ClickToCall and ClickToCallService share the configuration namespace '_genesys.widgets.clicktocall'. ClickToCall has UI options while ClickToCallService has connection options.

Example

```
{
        clicktocall: {

                'enableCountdown' : true,
                'provideAccessCode' : true,
                'autoDialAccessCode' : true,
                'ariaNumberExpirationIntervals' : [100, 75, 50, 25, 10],

                'ewt': {

                        'display': true,

                        'queue': 'QUEUE_NAME',

                        'threshold': 30,

                        'refreshInterval': 10
                },

                'confirmFormCloseEnabled' : true,

                'dataURL' : 'http://www.myphoneservice.org',

                'formJSON' : {},

                'userData' : {}
        }
}
```

Options

Name	Type	Description	Default	Required	Introduced / Updated
enableCountdown	boolean	Shows or hides the phone number expiration	true	false	

Name	Type	Description	Default	Required	Introduced / Updated
		counter.			
provideAccessCode	boolean	Enables or disables the use of a dial in access code for user verification.	true	false	
autoDialAccessCode	boolean	Enables or disables auto dialing the access code that is provided along with the dial in number.	true	false	
ariaNumberExpirationAnnouncements	array of boolean	An array containing the intervals in a percentage at which the screen reader will announce the remaining expiry time for the phone number. By default, it is enabled with the following time intervals, and is customizable according to user needs. Configuring a value of 'false' will let the screen reader read the phone number expiry time for every change.	[100, 75, 50, 25, 10]	false	9.0.016.11
ewt.display	boolean	Enables or disables showing of Estimated Wait Time (EWT) for an agent to be available.	true	false	
ewt.queue	string	A virtual queue name for the EWT API	none	Always required if EWT has to be	

Name	Type	Description	Default	Required	Introduced / Updated
		service channel.		displayed.	
ewt.threshold	integer	Display threshold for EWT in seconds. Does not display EWT if below threshold.	30	n/a	
ewt.refreshInterval	integer	EWT refresh interval in seconds.	10	n/a	
confirmFormCloseEnabled	boolean	Enable or disable displaying a confirmation message before closing ClickToCall, if information has been entered into the registration form.	true	n/a	
formJSON	object	An object representing the custom form to render. The definition placed here becomes the default registration form layout for ClickToCall. See Customizable ClickToCall Registration Form	A basic registration form is defined internally by default.	n/a	
userData	object	A custom object to be merged in with all network requests.	{}	n/a	

Localization

> **Important**
>
> For information on how to setup localization, please refer to the Localization section in the Deployment Guide

Usage

'clicktocall' namespace should be used when defining localization strings for ClickToCall 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.

Default i18n JSON

```
{
        "en": {
                "clicktocall": {
                        "Title": "ClickToCall",
                        "FirstName": "First Name",
                        "PlaceholderRequired": "Required",
                        "PlaceholderOptional": "Optional",
                        "LastName": "Last Name",
                        "PhoneNumber": "Phone",
                        "WaitTime": "Wait Time",
                        "FormCancel": "Cancel",
                        "AriaFormCancel": "Cancel",
                        "FormSubmit": "Request a number",
                        "AriaFormSubmit": "Request a number",
                        "PhoneLabel": "Dial in now",
                        "AriaPhoneTitle": "Opens the phone application",
                        "AccessLabel": "Access Code",
                        "ExpireLabel": "Number Expires in",
                        "AriaExpireLabel": "Number Expires in Timer",
                        "DisplayClose": "Close",
                        "AriaDisplayClose": "Close",
                        "NetworkFail": "Something went wrong, we apologize for the
inconvenience. Please check your connection settings and try again.",
                        "NetworkRetry": "OK",
                        "AriaNetworkRetry": "OK",
                        "InvalidAccept": "OK",
                        "AriaInvalidAccept": "OK",
```

```
                        "PhoneExpired": "Phone number has expired!",
                        "PhoneReRequest": "Request a new number",
                        "AriaPhoneReRequest": "Request a new number",
                        "LocalFormValidationEmptyPhoneNumber": "Please enter a phone number",
                        "ConfirmCloseWindow": "You have unsubmitted form data. Are you sure
you want to quit?",
                        "AriaConfirmCloseCancel": "No",
                        "ConfirmCloseCancel": "No",
                        "AriaConfirmCloseConfirm": "Yes",
                        "ConfirmCloseConfirm": "Yes",
                        "AriaWindowLabel": "Click To Call Window",
                        "AriaMaximize": "Click To Call Maximize",
                        "AriaMinimize": "Click To Call Minimize",
                        "AriaClose": "Click To Call Close"
                }
        }
}
```

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('ClickToCall.open');
```

configure

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

open

Opens the ClickToCall UI.

Example

```
oMyPlugin.command('ClickToCall.open', {

        userData: {},
        form: {

                autoSubmit: false,
                firstname: 'John',
                lastname: 'Smith',
                phonenumber: 9256349345
        },
        formJSON: {...}

}).done(function(e){
```

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

Options

Option	Type	Description
form	string	Object containing form data to prefill the ClickToCall form and optionally auto-submit the form.
form.autoSubmit	boolean	Automatically submit the ClickToCall form.
form.firstname	string	Value for the first name input entry field.
form.lastname	string	Value for the last input name entry field.
form.phonenumber	number	Value for the phone number input entry field.
formJSON	object	A JSON object containing a custom registration form definition. See Customizable ClickToCall Registration Form
userData	object	Arbitrary data that is attached with ClickToCall form submit request. Properties defined here are merged with default userData set in the configuration object.

Resolutions

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

close

Closes the ClickToCall UI.

Example

```
oMyPlugin.command('ClickToCall.close').done(function(e){

        // ClickToCall closed successfully

}).fail(function(e){

        // ClickToCall is already closed
});
```

Resolutions

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

minimize

Minimize or unminimize ClickToCall UI.

Example

```
oMyPlugin.command('ClickToCall.minimize').done(function(e){

        // ClickToCall minimized successfully

}).fail(function(e){

        // ClickToCall ignores command
});
```

Resolutions

Status	When	Returns
resolved	Always	n/a
rejected	Never	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('ClickToCall.ready', function(e){ /* sample code */ });
```

Name	Description	Data	Introduced / Updated
ready	ClickToCall is initialized and ready to accept commands.	API Metadata	
opened	The ClickToCall Widget has been opened.	API Metadata	
closed	The ClickToCall Widget has been closed.	API Metadata	
started	The user has started to fill out the ClickToCall form.	API Metadata	
cancelled	The user has stopped filling out the ClickToCall form and closed.	API Metadata	9.0.002.06
submitted	The user has submitted the form.	API Metadata	9.0.002.06
completed	The ClickToCall Widget form was filled out and a phone number requested.	API Metadata	
expired	The Phone number ClickToCall widget requested has expired.	API Metadata	
minimized	The ClickToCall widget has been changed to a minimized state.	n/a	

Name	Description	Data	Introduced / Updated
unminimized	The ClickToCall widget has been restored from a minimized state to the standard view.	n/a	

Metadata

Interaction Lifecycle

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

The following events are part of the Interaction Lifecycle:

```
ready
opened
started
submitted
cancelled
completed
closed
```

Lifecycle Scenarios

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

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

```
ready -> opened -> cancelled -> closed
```

The user started filling out the form but closed ClickToCall without submitting the phone number request:

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

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

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

> **Tip**
> For a list of all ClickToCall 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 ClickToCall 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 ClickToCall interactions. During run-time, the metadata can help you offer a smart and dynamic experience to your users.

Reference

Name	Type	Description	Introduced / Updated
ready	integer (timestamp)	Timestamp indicating when ClickToCall is ready.	
opened	integer (timestamp)	Timestamp indicating when ClickToCall was opened.	
started	integer (timestamp)	Timestamp indicating when user started filling out the form.	
cancelled	integer (timestamp)	Timestamp indicating when the ClickToCall was cancelled without a phone request.	
completed	integer (timestamp)	Timestamp indicating when ClickToCall successfully filled out and submitted a form.	
closed	integer (timestamp)	Timestamp indicating when ClickToCall was closed.	
expired	integer (timestamp)	Timestamp indicating when the requested phone number expired.	
form	object	An object containing the form parameters when the form is submitted.	9.0.002.06

Customizable ClickToCall Registration Form

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

- Text
- Select
- Hidden
- Checkbox
- Textarea

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

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

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

Default Example

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

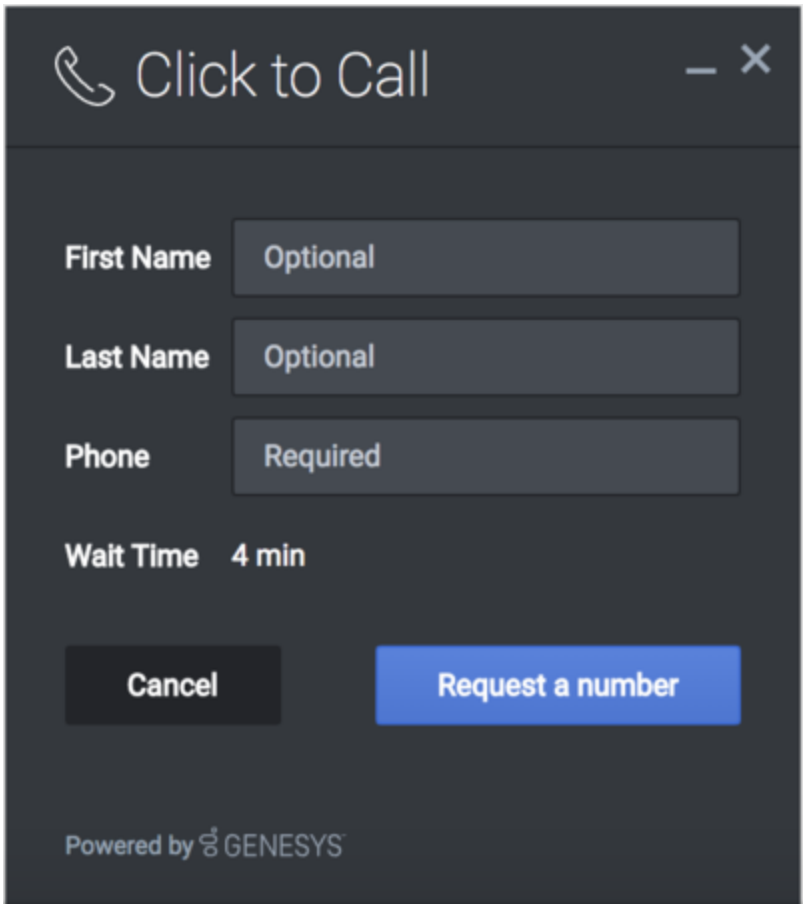
> ### Important
> You can define *any* number of inputs here, of *any* supported type, in *any* combination. Our example below simply demonstrates how WebChat defines its default form internally.

> ### Important
> The Phone Number field with name "phonenumber" is required for all custom ClickToCall forms. This field value is needed to request a phone number from the Genesys callback API.

```
formJSON : {
                                wrapper: "<table>",
```

```
                                inputs: [

                                {
                                        id: "cx_clicktocall_form_firstname",
                                        name: "firstname",
                                        maxlength: "100",
                                        placeholder: "@i18n:clicktocall.PlaceholderOptional",
                                        label: "@i18n:clicktocall.FirstName"
                                },

                                {
                                        id: "cx_clicktocall_form_lastname",
                                        name: "lastname",
                                        maxlength: "100",
                                        placeholder: "@i18n:clicktocall.PlaceholderOptional",
                                        label: "@i18n:clicktocall.LastName"
                                },

                                {
                                        id: "cx_clicktocall_form_phonenumber",
                                        name: "phonenumber",
                                        maxlength: "100",
                                        placeholder: "@i18n:clicktocall.PlaceholderRequired",
                                        label: "@i18n:clicktocall.PhoneNumber",

                                        onkeypress: function(event) {
                                                return (event.charCode >= 48 &&
event.charCode <= 57) || (event.charCode == 43);
                                        }
                                }
                                ]
                        }
```

Using this definition will result in this output:



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

Property	Type	Default	Description
			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.myCusto For more information, see the Labels section.
wrapper	HTML string	"<tr><th>{label}</th><td>{input}</td></tr>"	Each input exists in its own row in the form. By default this is a table-row with the label in the left cell and the input in the right cell. You can redefine this wrapper and layout by specifying a new HTML row structure. See the Wrappers section for more info. The default wrapper for an input is "<tr><th>{label}</th><td>{input}<
validate	function		Define a validation function for the input that executes when the input loses focus (blur) or changes value. Your function must return `true` or `false`. True to indicate it passed, false to indicate it failed. If your validation fails, the form will not submit and the invalid input will be highlighted in red. See the Validation section for more details and examples.
validateWhileTyping	boolean	false	Execute validation on keypress in addition to blur and change. This

Property	Type	Default	Description
			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, {text: ‘Option 1’, value: ‘1’} for a selectable option, and {text: "Group 1", group: true} for an option group).

HTML Attributes

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

Example

```
{
        id: "cx_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:clicktocall.PlaceholderOptional",
        label: "@i18n:clicktocall.FirstName"
}
```

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_form_firstname
 name="firstname" maxlength="100" placeholder="Optional"></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 "<table></table>". This is the default wrapper for the ClickToCall 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_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:clicktocall.PlaceholderOptional",
        label: "@i18n:clicktocall.FirstName",
        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_form_firstname",
        name: "firstname",
        maxlength: "100",
        placeholder: "@i18n:clicktocall.PlaceholderOptional",
        label: "@i18n:clicktocall.FirstName",

        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 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 element being validated.
label	HTML reference	A jquery reference to the label for the input being validated.
$	jquery instance	Widget's internal jquery instance. Use this to help you write your validation logic, if needed.
CXBus	CXBus instance	Widget's internal CXBus reference. Use this to call commands on the bus, if needed.
Common	Function Library	Widget's internal Common library of functions and utilities. Use if needed.

Form Submit

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

Please note the Phone Number field - with 'name: "phonenumber"', this field is required for all custom ClickToCall forms as this field value is needed to request a phone number from the Genesys Callback API.

Form Prefill

You can prefill the custom form using ClickToCall.open command by passing the form (form data) and formJSON (custom registration form), provided the form input names in the formJSON must match with the property names in the form data.

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

```
_genesys.widgets.bus.command("ClickToCall.open", {

        formJSON: {
                   wrapper: "<table>",
                    inputs: [{
                            id: "cx_form_phone_number",
                            name: "phonenumber",
                            maxlength: "12",
                            placeholder: "@i18n:clicktocall.PlaceholderRequired",
                            label: "@i18n:clicktocall.PhoneNumber"
                        }]
        },

        form: {
                phonenumber: 9453222222
        }
});
```

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.config(object)

Configure some debug options for Common at runtime.

Example

'Enable full debug logging'

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

Arguments

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

Common.checkPath(object, path)

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

Example

'Check for window._genesys.main'

```
var oMainConfig = false;

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

Arguments

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

Argument	Type	Description
		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.

Argument	Type	Description
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 displayed.

Common.insertIcon

Adds an icon before the selected element.

Example

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

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

Arguments

Argument	Type	Description
element	jQuery DOM selector	An html element to which Icon is to be displayed.
Icon name	string	Name of the Icon that you would like to display. Note: Refer to Common.debugIcons method to find out all the icons names that widgets supports.
Icon Aria Name	string	Name for the Icon to be read by

Argument	Type	Description
		screen readers.

Common.injectScript

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

Example

'Inject your Widget WebChat extension plugin.'

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

Arguments

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

Common.mobileScreenScale

Re-sizes and fits Widget to any mobile screen.

Example

'Fit your widget to any mobile screen.'

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

Arguments

Argument	Type	Description
element	jQuery DOM Selector	Your main Widget wrapper container selector that contains

Argument	Type	Description
		the entire Widget with 'cx-titlebar', 'cx-body', 'cx-footer', 'cx-button-container' and 'cx-message-container' classes in it.

Common.showLoading

Show loading spinner Icon.

Example

'Show loading spinner during an Ajax request'

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

Arguments

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

Common.hideLoading

Remove loading spinner Icon.

Example

'Remove loading spinner after the Ajax request'

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

Arguments

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

Common.showWaiting

Show waiting Icon.

Example

'Show waiting Icon when uploading a file.'

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

Arguments

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

Common.hideWaiting

Remove waiting Icon.

Example

'Remove waiting Icon after file upload is done.'

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

Arguments

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

Common.watch

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

Example

'Make Request Notifications till none are pending.'

```
Common.watch(function(iteration, maxIterations){

        if(bRequestNotificationsPending){
         // ..POST Request
        }
        return !bRequestNotificationsPending;
}, 3000, 30)
```

Arguments

Argument	Type	Description
function name	function	The function that you would like to execute. It should return true/ false.
frequency	milliseconds	Execute the function for every 'x' milliseconds 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.

Argument	Type	Description
name	string	The actual dialog box name.

Common.hideDialog

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

Example

'Hide dialog box'

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

Arguments

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

Common.hideDialogs

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

Example

'Hide all dialog boxes.'

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

Arguments

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

Common.showAlert

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

Example

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

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

Arguments

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

Common.bytesToSize

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

Example

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

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

Arguments

Argument	Type	Description
bytes	number	Number in bytes size.

Common.getFormattedTime

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

Example

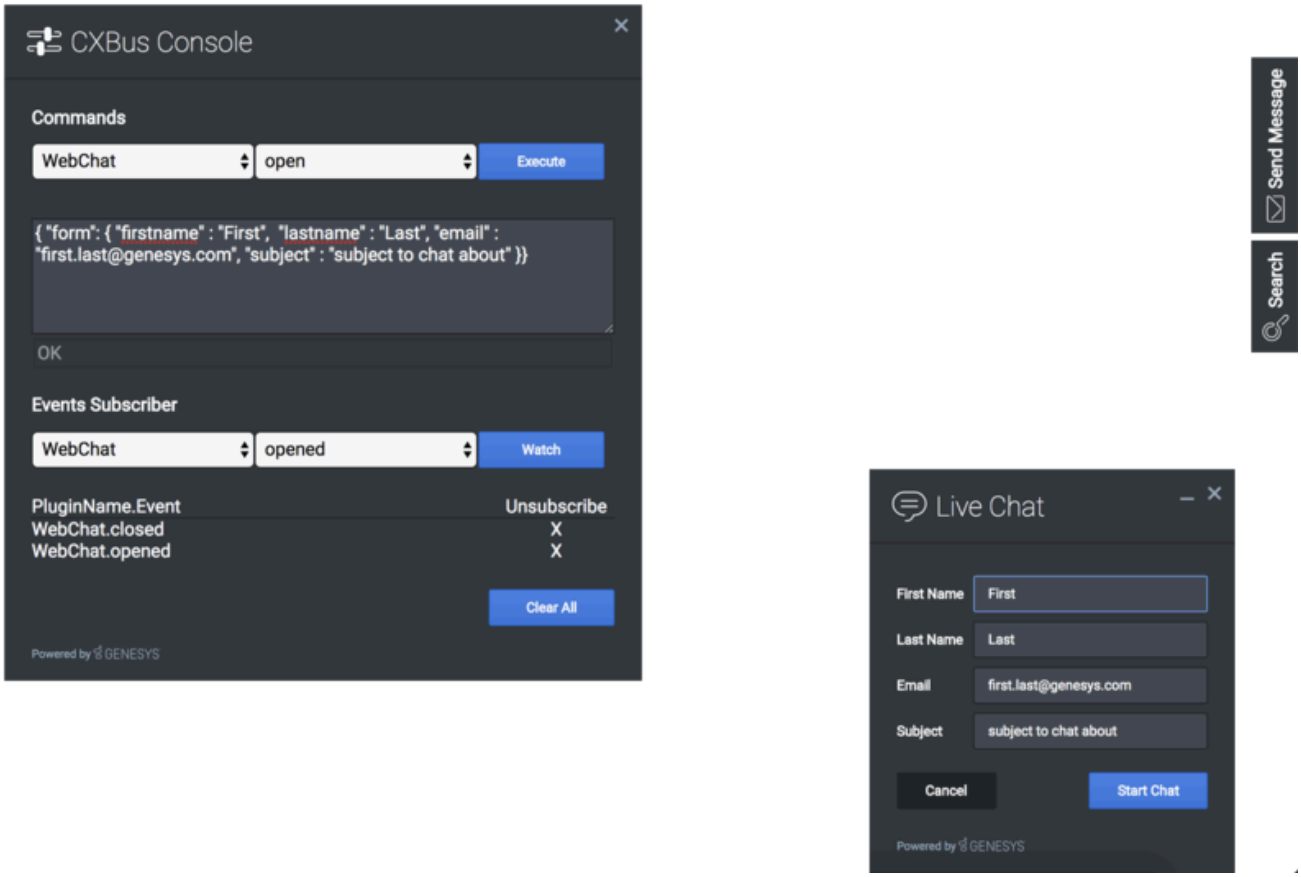
'convert date timestamp to return time in 12 hrs format'

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

Arguments

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

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 set up 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.

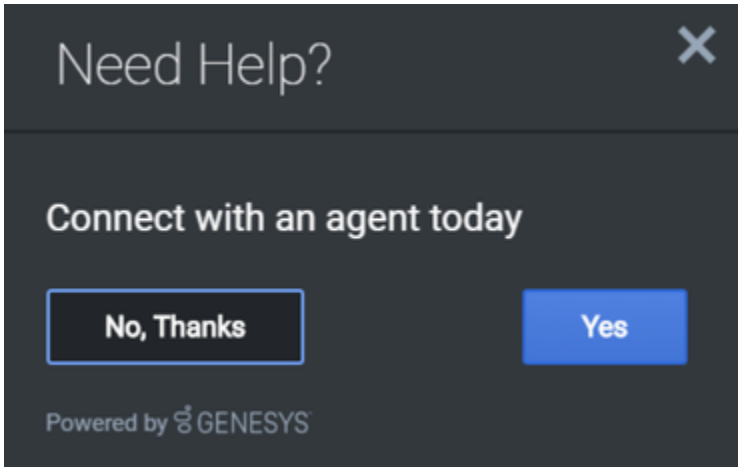
> **Important**
>
> The global bus object is a debug tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Widgets Extensions for more information about extending Genesys Widgets.

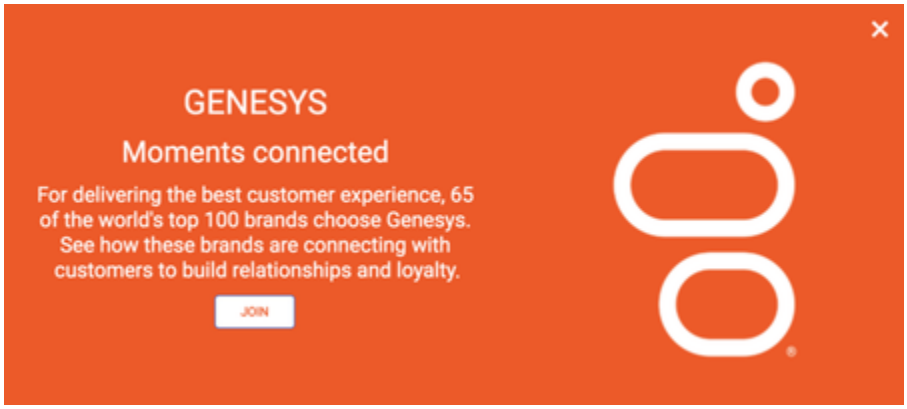
```
var oMyPlugin = window._genesys.widgets.bus.registerPlugin('MyPlugin');

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

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

Engage

Introduced: 9.0.002.06





- Configuration
- Localization
- API Commands
- API Events

Overview

The Engage plugin can integrate any Engage solution with Widgets. This plugin is generic and contains commands that automate engagement within Widgets. Starting with version 9.0.015.11, the Engage plugin includes Offers, which allows a customer to view a product or promotion on a page. It

comes with many display modes and rendering options, such as overlay/toaster mode with text or image-only layouts, or both.

Usage

The Engage plugin can be used to show either an invite or an offer via the following methods:

- Calling the Engage.invite command
- Calling the Engage.offer command

Namespace

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

Type	Namespace
i18n - Localization	Engage
CXBus - API Commands & API Events	Engage
CSS	.cx-engage

Screenshots

Engage Invite

- 

  Mobile mode Engage Invite view with dark theme

- 

Mobile mode Engage Invite view with light theme

Engage Offer

- 

Desktop Toast view with both image and text

- 

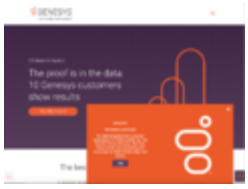
Desktop Modal Overlay view with text on top

- 

Desktop Overlay view with text at bottom

- 

Desktop Toast view with text content on right side

- 

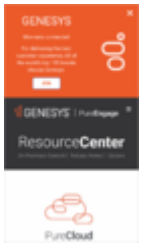
Desktop Toast view with text content on left side

- 

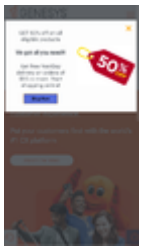
Desktop Overlay view with text on right side

- 

Desktop Modal Overlay view with text on left side

- 

Mobile Offer inserted onto the top of a web page

- 

Mobile Offer view in modal overlay with background area grayed out

Mobile Offer view in modal overlay

Configuration

Description

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('Engage.invite');
```

invite

Opens the Engage Widget and renders the text based on the options provided. If no options are provided, it will not open.

Example

```
oMyPlugin.command('Engage.invite', {
        'type':'toast',
        'timeout':3000,
        'title':'Engage Title',
        'ariaTitle':'Engage Invite',
        'body':'Engage invite body content',
        'accept':'Yes',
        'decline':'No, thanks',
        'ariaAccept':'Yes',
        'ariaDecline':'No, thanks',
        'ariaClose':'Close',
        'command': 'WebChat.open',
        'options':{'proactive': true, 'userData': {'category': 'shoes'}}
});

oMyPlugin.command('Engage.invite',{
        'type':'toast',
        'timeout':3000,
        'force': true,
        'title':'Engage Title',
        'ariaTitle':'Engage Invite',
        'body':'Engage invite body content',
```

```
        'accept':'Yes',
        'decline':'No, thanks',
        'ariaAccept':'Yes',
        'ariaDecline':'No, thanks',
        'ariaClose':'Close'
}).done(function(response){

        // Act upon the received response code

        switch(response){
        case 'accepted':oMyPlugin.command('WebChat.open');
                        break;
        case 'declined': break;
        case 'closed': break;
        case 'timeout': break;
}

});
```

Options

Option	Type	Description	Accepted Values	Default	Introduced / Updated
type	string	Widget display type.	toast		
timeout	number	Timeout integer in milliseconds.	n/a		
title	string	String for widget title.	n/a		
ariaTitle	string	Aria label text for the Engage invite window.	n/a		9.0.015.04
body	string	String for offer body text.	n/a		
accept	string	String for Accept button text.	n/a		
ariaAccept	string	Aria label text for the Accept button.	n/a		9.0.016.10
decline	string	String for Decline button text.	n/a		
ariaDecline	string	Aria label text for the Decline button.	n/a		9.0.016.10
ariaClose	string	Aria label text for the Engage Close button.	n/a		9.0.016.10

Option	Type	Description	Accepted Values	Default	Introduced / Updated
command	string	Command to execute.	n/a		
options	object	Options related to the command provided.	n/a		
priority	number	Replace the active lower priority Engage invite with the higher priority Engage invite.	n/a	0	9.0.015.11
force	boolean	Replace the active Engage invite with the new Engage invite irrespective of priorities.	n/a	false	9.0.015.11

Resolutions

Status	When	Returns
resolved	When engage invite is accepted by user.	accepted
resolved	When engage invite is declined by user.	declined
resolved	When engage invite widget is closed by user.	closed
resolved	When engage invite widget closes due to timeout.	timeout

offer

Opens an Offer Widget using the data sent through the command options provided below. It can include both rendering options and the actual data that needs to be displayed in the Offer Widget. If no options are provided, it will not open.

Example

```
oMyPlugin.command('Engage.offer', {
```

```
        mode:'overlay',
        modal:true,
        layout:'leftText',
        priority: 1,
        title: 'GRAB WHAT YOU NEED!!',
        ariaTitle: 'Offers',
        headline:'We Got All!',
        description:'Get free NextDay delivery on orders of $35 or more. Start shopping now!',

        cta:{
                text:'Join',
                url:'https://www.genesys.com',
                target:'_blank'
        },

        image:{
                src:'https://picsum.photos/id/237/300/300',
                alt:'Alternate Text for Image'
        },

        styles:{
                closeButton:{
                        'color':'red'
                }
        },
        ariaCTA:'Join',
        ariaClose:'Close Offer'
});
```

Options

Option	Type	Description	Accepted Values	Default	Introduced / Updated
mode	string	Display type of Offer Widget.	overlay, toaster	toaster	9.0.015.04
modal	boolean	Applicable only when mode is 'overlay'. A smokescreen will be shown in the background of overlay modal window. This window can be dismissed by clicking anywhere in the smokescreen area.	n/a	false	9.0.015.04
layout	string	Additional layout options supported for all modes.	minimal, leftText, rightText, topText,	leftText	9.0.015.04

Option	Type	Description	Accepted Values	Default	Introduced / Updated
			bottomText		
headline	string	Offer title header text.	n/a	n/a	9.0.015.04
ariaTitle	string	Aria label text for the Offer window.	n/a	n/a	9.0.015.04
description	string	Offer body description text.	n/a	n/a	9.0.015.04
cta	object	An object containing html attributes and/or CXBus command for the CTA (call to action) button.	n/a	n/a	9.0.015.04
cta.text	string	CTA button text.	n/a	n/a	9.0.015.04
cta.url	string	URL string for the CTA button. `Note: The URL must be properly defined with the complete Protocol URL Address. For example, https://www.genesys.com.`	_blank, _parent, _self, _top, framename	n/a	9.0.015.04
cta.target	string	To specify where the URL should be opened.	n/a	n/a	9.0.015.04
cta.command	string	A CXBus command to execute.	n/a	n/a	9.0.015.04
cta.commandOptions	string	Options related to CXBUs command.	n/a	n/a	9.0.015.04
image	object	An object containing image tag attributes.	n/a	n/a	9.0.015.04
image.src	string	URL of the image.	n/a	n/a	9.0.015.04
image.alt	string	Alternate text for the image.	n/a	n/a	9.0.015.04

Option	Type	Description	Accepted Values	Default	Introduced / Updated
image.title	string	To indicate the screen reader user whether the image opens the URL in a new window.	n/a	n/a	9.0.016.10
insertAfter	string	Applicable only in mobile mode. An id or class name of an html selector from the host page. The Offer will be inserted after this element. The value mentioned here should be preceded with the standard Class ('.') and ID selector ('#') character.	n/a	n/a	9.0.015.04
insertBefore	string	Applicable only in mobile mode. An id or class name of an html selector from the host page. The Offer will be inserted before this element. The value mentioned here should be preceded with the standard Class ('.') and ID selector ('#') character.	n/a	n/a	9.0.015.04
insertInto	string	Applicable only in mobile mode. An id or class name of an html selector from the host page. The Offer will	n/a	n/a	9.0.015.04

Option	Type	Description	Accepted Values	Default	Introduced / Updated
		be appended inside this element. The value mentioned here should be preceded with the standard Class ('.') and ID selector ('#') character.			
styles	object	An Object containing styles for the Offer content.	n/a	n/a	9.0.015.04
styles.closeButton	object	An Object containing styles for the close button.	n/a	n/a	9.0.015.04
styles.closeButton.color	string	Color of the close button.	n/a	n/a	9.0.015.04
styles.closeButton.opacity	number	CSS 'opacity' property for the close button.	n/a	n/a	9.0.015.04
styles.overlay	object	An Object containing styles for the overlay container.	n/a	n/a	9.0.015.04
styles.overlay.top	string	CSS 'top' property for the overlay container.	n/a	n/a	9.0.015.04
styles.overlay.right	string	CSS 'right' property for the overlay container.	n/a	n/a	9.0.015.04
styles.overlay.bottom	string	CSS 'bottom' property for the overlay container.	n/a	n/a	9.0.015.04
styles.overlay.left	string	CSS 'left' property for the overlay container. `Note: When all the position`	n/a	n/a	9.0.015.04

Option	Type	Description	Accepted Values	Default	Introduced / Updated
		`values are provided, the order of precedence will be Top, Right, Bottom and Left.`			
styles.overlay.center	boolean	Aligns overlay container to the center of the screen.	n/a	true	9.0.015.04
styles.offer	object	An Object containing styles for the Offer window.	n/a	n/a	9.0.015.04
styles.offer.backgroundColor	string	Background color of the Offer.	n/a	n/a	9.0.015.04
styles.offer.color	string	Text color of the Offer.	n/a	n/a	9.0.015.04
styles.offer.padding	string	Padding for the Offer container.	n/a	0	9.0.015.04
styles.title	object	An Object containing styles for the title.	n/a	n/a	9.0.015.04
styles.title.font	string	CSS 'font' property for the title.	n/a	n/a	9.0.015.04
styles.title.textAlign	string	CSS 'text-align' property for the title.	n/a	n/a	9.0.015.04
styles.headline	object	An Object containing styles for the header text.	n/a	n/a	9.0.015.04
styles.headline.font	string	CSS 'font' property for the header text.	n/a	n/a	9.0.015.04
styles.headline.textAlign	string	CSS 'text-align' property for the header text.	n/a	n/a	9.0.015.04
styles.description	object	An Object containing styles for the	n/a	n/a	9.0.015.04

Option	Type	Description	Accepted Values	Default	Introduced / Updated
		Offer description text.			
styles.description.font	string	CSS 'font' property for the description text.	n/a	n/a	9.0.015.04
styles.description.textAlign	string	CSS 'text-align' property for the description text.	n/a	n/a	9.0.015.04
styles.ctaButton	object	An Object containing styles for call to action button in the Offer window.	n/a	n/a	9.0.015.04
styles.ctaButton.font	string	CSS 'font' property for the text in call to action button.	n/a	n/a	9.0.015.04
styles.ctaButton.textAlign	string	CSS 'text-align' property for the text in call to action button.	n/a	n/a	9.0.015.04
styles.ctaButton.background	string	CSS 'background' property for the call to action button.	n/a	n/a	9.0.015.04
styles.ctaButton.color	string	CSS 'color' property for the text in call to action button.	n/a	n/a	9.0.015.04
styles.ctaButton.fontSize	string	CSS 'font-size' property for the text in call to action button.	n/a	n/a	9.0.015.04
ariaCTA	string	Aria label text for the Offer CTA button.	n/a	n/a	9.0.016.10
ariaClose	string	Aria label text for the Offer Close button.	n/a	n/a	9.0.016.10
priority	number	Replace the	n/a	0	9.0.015.11

Option	Type	Description	Accepted Values	Default	Introduced / Updated
		active lower priority Engage Offer with the higher priority Engage Offer.			
force	boolean	Replace the active Engage Offer with the new Engage Offer irrespective of priorities.	n/a	false	9.0.015.11

API Events

Once you've registered your 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('Engage.ready', function(e){});
```

Name	Description	Data	Introduced / Updated
ready	The Engage Widget is initialized and ready to accept commands on the bus.	n/a	
opened	The Engage Widget has opened. `Note: Applicable only to Engage.offer command`	Metadata	9.0.015.04
CTA	When the user clicks on the CTA button in Engage Widget. `Note: Applicable only to Engage.offer command`	Metadata	9.0.015.04
hover	When the user first hovers over the Engage Widget. `Note: Applicable only to Engage.offer command`	Metadata	9.0.015.04
dismissed	When the user closes the Engage Widget by	Metadata	9.0.015.04

Name	Description	Data	Introduced / Updated
	clicking on the close button. `Note: Applicable only to Engage.offer command`		
closed	The Engage Widget has closed. `Note: Applicable only to Engage.offer command`	Metadata	9.0.015.04

Metadata

> **Important**
> Applicable only for Engage.offer command

Interaction Lifecycle

Every Offer Engage 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 Engage Offers), to the end (closing Offers), and every step in between.

The following events are part of the Interaction Lifecycle:

```
ready
opened
CTA
hover
dismissed
closed
```

Lifecycle Scenarios

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

The user opened the Offer Engage Widget but changed their mind and closed it without seeing the Offer details:

```
ready -> opened -> dismissed -> closed
```

The user opened the Offer Engage Widget, hovered over Offer details then closed it:

```
ready -> opened -> hover -> dismissed -> closed
```

The user opened the Offer Engage Widget and clicked on the button, which triggers CTA:

```
ready -> opened -> CTA -> closed
```

> **Tip**
> For a list of all Offer Engage events, see API Events.

Metadata

Each event in the Interaction Lifecycle includes the following block of metadata. By default, all values are set to `false`. As the user progresses through the lifecycle of an Offer Engage interaction, these values are updated.

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

Reference

Name	Type	Description	Introduced / Updated
opened	integer (timestamp)	Timestamp indicating when the Offer was opened.	9.0.015.04
closed	integer (timestamp)	Timestamp indicating when the Offer was closed.	9.0.015.04
dismissed	integer (timestamp)	Timestamp indicating when the user dismissed the Offer by clicking the close button.	9.0.015.04
triggeredCTA	integer (timestamp)	Timestamp indicating when the CTA was triggered.	9.0.015.04
timeBeforeCTA	integer (milliseconds)	Total time in milliseconds from when the user opened the Offer to when the CTA is triggered.	9.0.015.04
timeFirstHover	integer (timestamp)	Timestamp indicating when the user first hovered over Offer.	9.0.015.04
timeBeforeHover	integer (milliseconds)	Total time in milliseconds from when the user opened the Offer to when the user first hovered over Offer.	9.0.015.04
timeElapsedHover	integer (milliseconds)	Total time in milliseconds when the user hovered over Offer.	9.0.015.04
elementClicked	string	Name of CTA element that was clicked ('button').	9.0.015.04

KnowledgeCenterService

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.widgets.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.widgets.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 knowledge bases that are publicly available.		Always
lang	string	Language in which knowledge search will be executed.	en	Always
enableHTML	boolean	By default, articles are fetched in plain-text. To enable HTML-formatted articles,	false	

Name	Type	Description	Default	Required
		set this value to true.		
media	string	Media that content needs to be searched for. Empty value allows any available content to be searched.	all	
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).	widgets	
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	
apiVersion	string	Knowledge Center Server API Version. 'v2' for 9.0 Knowledge Center, 'v1' for 8.5 Knowledge Center. (since 9.0.002.06)	v2	

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
apiVersion	string	Knowledge Center Server API Version. 'v2' for 9.0 Knowledge Center, 'v1' for 8.5 Knowledge Center.

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 trending 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

Option	Type	Description
		items
categories	object	Array of Category IDs for 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'

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
kbId	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 its parameters had changed.	{sessionId: (string), language: (string), media: (string), customer: (string)}
noResultsFound	Search did not return results or results are empty	{query: (string), language: (string), media: (string), knowledgebases: (list of string), sessionId: (string), tenantId: (string)}
documentOpened	Full document content has been requested	{language: (string), knowledgebase: (string), sessionId: (string), tenantId: (string), documentId: (string), document: (object)}
searched	Query has been searched and results provided	{query: (string), language: (string), media: (string), knowledgebases: (list of strings), sessionId: (string), documents: (list of objects)}
suggested	Autocomplete queries have been suggested	{language: (string), sessionId: (string), tenantId: (string), query: (string), categories: (list of

Name	Description	Data
		objects), filters: (object), media: (string), knowledgebases: (list of strings)}
voted	Relevancy feedback has been provided for the search result	{language: (string), sessionId: (string), tenantId: (string), relevant: (boolean), knowledgebase: (string), documentId: (string)}

Overlay

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 moved between different container types.

Overlay provides these benefits:

- Shows UI in center of window.
- Opens and closes 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 it all to Overlay.open.

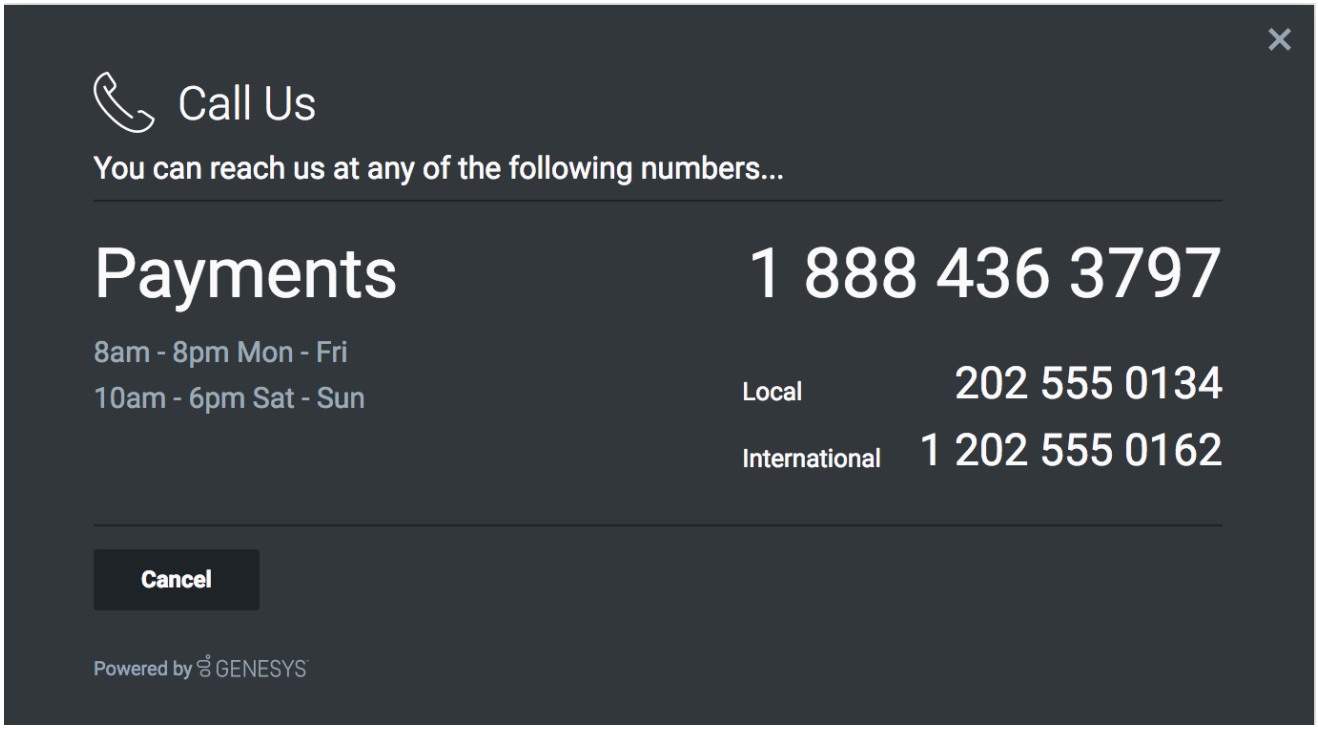
Customization

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	The name of the overlay window group you want to add a new overlay view into.

Resolutions

Status	When	Returns
resolved	When overlay is successfully opened	{html: <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	n/a

Status	When	Returns
	closed.	
rejected	When Overlay is already closed.	'Overlay view is already closed'
rejected	When Overlay view is immutable.	'Overlay view is currently reserved'

API Events

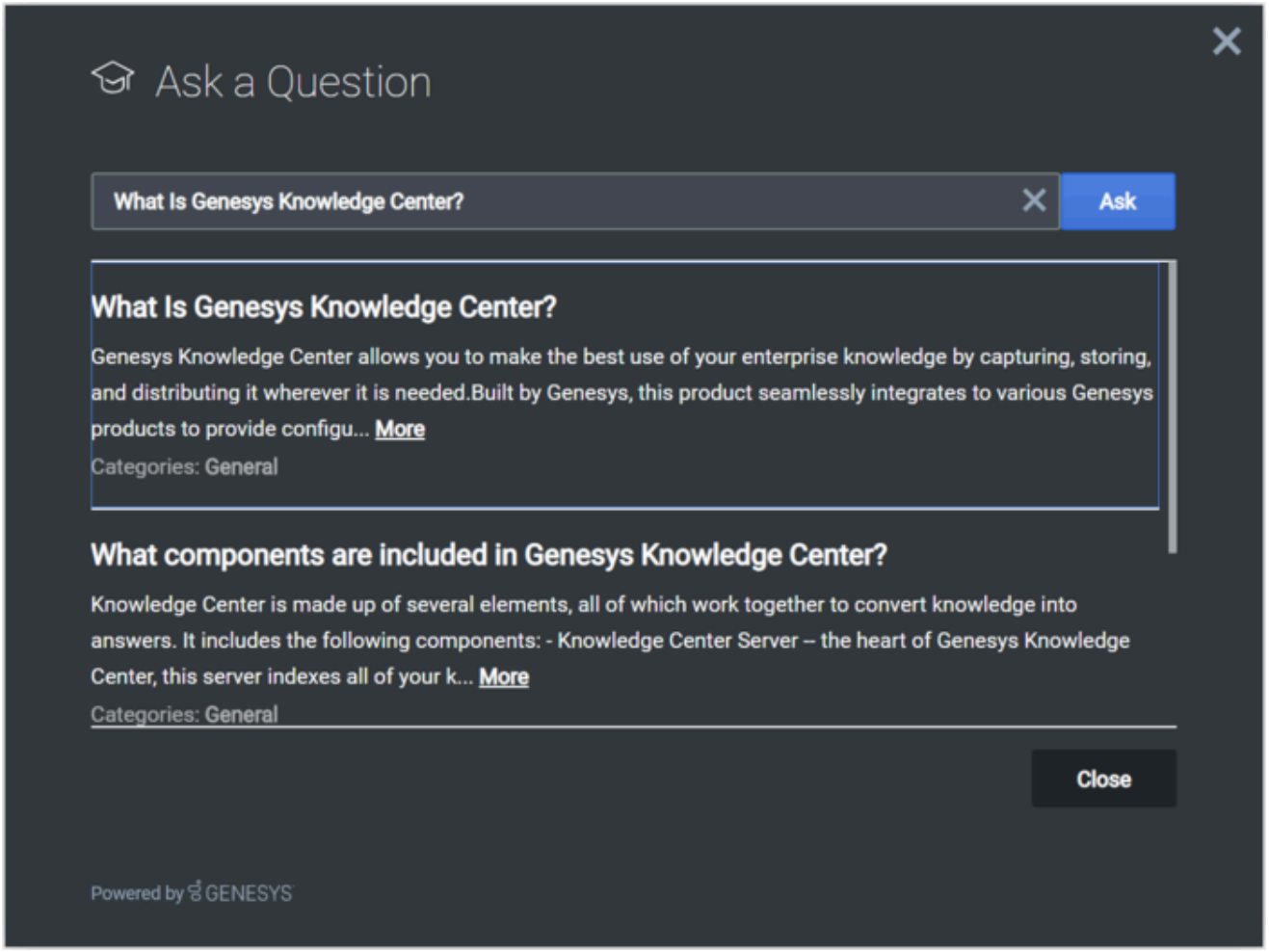
Once you've registered your own plugin on the bus, you can subscribe and listen for published events. Below we'll quickly register a new plugin on the bus using the global bus object.

> **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.

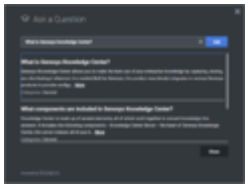
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

Screenshots

"Dark" Theme

- 

Desktop Search Widget with contextual help when typing

- 

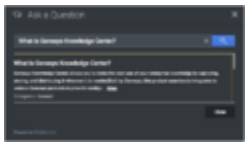
  Desktop Search Widget showing search results

- 

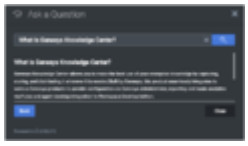
  Desktop Search Widget showing document details

- 

  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. Note: In case of running Widgets in lazy load mode, this option requires Search plugin to be pre-loaded	false	
SearchButton.template	string	Custom HTML string template for search button.	<span	

Name	Type	Description	Default	Required
			`class='cx-icon' data-icon='search'></span><span class='i18n cx-search-button-label' data-message='SearchButton'></span>`	
SearchButton.effect	string	Type of animation effect when revealing chat button. 'slide' or 'fade'.	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!",
                        "GKCIsUnavailable": "Knowledge Center Server is currently not
```

```
available.",
                        "AriaAsk": "Ask",
                        "AriaClose": "Search Close",
                        "AriaYes": "Yes",
                        "AriaNo": "No",
                        "AriaBack": "Back to the Search Results",
                        "AriaClear": "Clear the Search Text",
                        "AriaSearch": "Search",
                        "AriaWindowLabel": "Search Window",
                        "AriaSearchDropdown": "Suggested results",
                        "AriaSearchMore": "Read more about",
                        "AriaResultsCount": "Total number of results"
                }
        }
}
```

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 or search button is disabled in configuration
});
```

Options

Option	Type	Description
openDelay	number	Duration in milliseconds to delay showing the search buton 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 and	n/a

Status	When	Returns
	currently not visible.	
rejected	When the search button is either not enabled in the configuration, or it's already visible	'Search button is not enabled in the configuration, or already visible. Ignoring command.'

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.

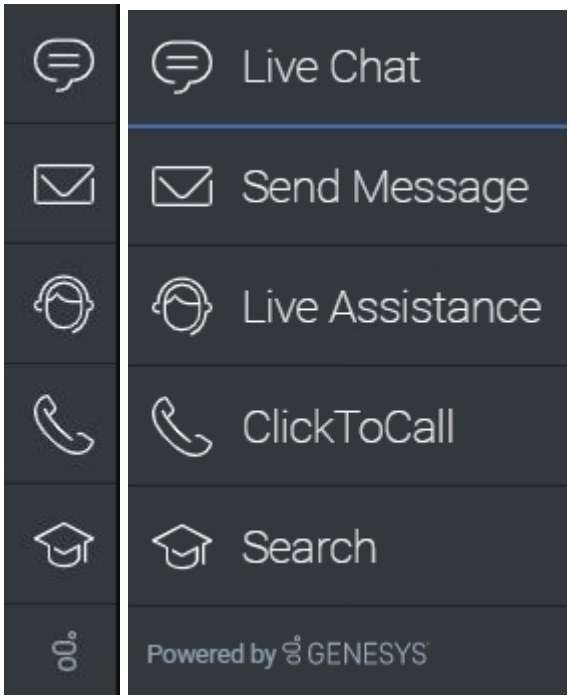
> **Important**
>
> The global bus object is a debug tool. When implementing Widgets on your own site, do not use the global bus object to register your custom plugins. Instead, see Widgets Extensions for more information about extending Genesys Widgets.

```
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.

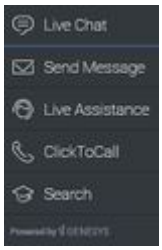
Switching between desktop and mobile mode is done automatically by default. You may configure Genesys Widgets to switch between Desktop and Mobile mode manually if necessary.

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

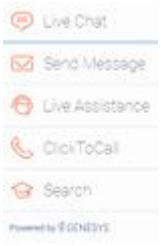
- 

  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',
                        clickOptions: {},

                         //use your own static string or i18n query string for the below two
display properties
                        displayName: 'Live Assist',
                        displayTitle: 'Get live help',

                        icon: 'agent'
                },

                {
                        name: 'Search',
                        clickCommand: 'Search.open',
                        clickOptions: {},

                         // Example of i18n query string: '@i18n:search.SearchName' where
'search' refers to the plugin namepsace 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');
                        }
                },

                {
```

```
                        name: 'Offers',
                        displayName: '@i18n:sidebar.OffersName',
                        displayTitle: '@i18n:sidebar.OffersName'
                },

                {
                        name: 'ClickToCall',
                        displayName: '@i18n:sidebar.ClickToCallName',
                        displayTitle: '@i18n:sidebar.ClickToCallTitle'
                },

                {
                        name: 'WebChat'
                }
        ]
};
```

Options

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

Name	Type	Description	Default	Required
		default values or when defining a new custom channel/plugin, use the below following properties.		
channels[index].clickCommand	string	Change the default command that is triggered when clicked.	n/a	false
channels[index].clickOptions	object	Pass valid command options that are used in above click command execution.	n/a	n/a
channels[index].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. For the list of Icon names see Included Icons.	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

Example i18n JSON

```
{
        "en": {
                "sidebar": {
                        "SidebarTitle": "Need help?",
                        "ChannelSelectorName": "Live Assistance",
                        "ChannelSelectorTitle": "Get assistance from one of our agents right
away",
                        "SearchName": "Search",
                        "SearchTitle": "Search",
                        "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",
                        "ClickToCallName": "Click To Call",
                        "ClickToCallTitle": "Request a customer service phone number",
                        "AriaClose": "Close the menu Need help"
                }
        }
}
```

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
                        icon: 'agent',
                        onClick: function($, CXBus, Common) {
```

```
                                _genesys.widgets.bus.command('MyPlugin.open');
                        }
                }
                ...
        ]

}).done(function(e){

        // Sidebar configured successfully

}).fail(function(e){

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

Options

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

Option	Type	Description
		content for this channel with your own static string or to achieve localization, use i18n query string. Syntax: @i18n:<plugin namespace>.<display key>.
channels[index].icon	string	Change the default Icon for this channel. For the list of Icon names see Included Icons.
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 channel icon and name.	n/a
contracted	Sidebar widget has contracted back, showing channel icons only.	n/a

StatsService

Overview

StatsService exposes a high-level API for utilizing Genesys Stats services. You can use these services to fetch estimated wait 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 Wait 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.widgets.stats'. StatsService has connection settings to fetch EWT details from each channel.

Example

```
window._genesys.widgets.stats = {

        ajaxTimeout: 3000,

        ewt: {

                dataURL: 'http://10.0.0.121:7777/genesys/1/service/ewt-for-vq',
                apikey: 'n3exxxxxXREBMYjGxxxx8VA',
                apiVersion: 'v1',
                mode: 'urs2'
        }
};
```

Options

Name	Type	Description	Default	Required	Accepted Values
ajaxTimeout	number	Number of milliseconds to wait before AJAX timeout	3000	n/a	n/a
ewt.apikey	string	Apigee Proxy secure token. If apiVersion is v3, this holds the x-api-key value.	n/a	Yes, if using Apigee Proxy. or v3 API.	n/a
ewt.dataURL	URL String	URL to the API endpoint for Estimated Wait Time (EWT)	n/a	Always	n/a
ewt.apiVersion	string	Version of EWT API.	'v1'	Yes, if using GMS EWT v2 or EWT v3 dataURL	'v1', 'v2', 'v3'

Name	Type	Description	Default	Required	Accepted Values
		Note: This value determines the version of EWT API in GMS/v3. That is: 'v1' - GMS EWT v1 'v2' - GMS EWT v2 'v3' - EWT v3 Only GET request type with virtual queue name as query parameters are supported.			
ewt.mode	string	EWT mode parameter for GMS/v3 API. This value will vary based on the above apiVersion.	Will vary based on the above apiVersion as shown below. 'urs2' for 'v1' 'ewt2' for 'v2' 'mode2' for 'v3'	n/a	'urs','urs2' or 'stat' for 'v1' 'ewt1,'ewt2' or 'ewt3' for 'v2' 'mode1','mode2' or 'mode3' for 'v3'

Localization

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.getStats');
```

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',
        mode: 'urs2'

}).done(function(e){

        // StatsService got stats successfully

}).fail(function(e){

        // StatsService failed to get stats
});
```

Options

Option	Type	Description
group	string	Mention specific group name you would like to request like EWT,

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

Resolutions

Status	When	Returns
resolved	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'

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 AJAX Response data
error.ewt	An error occurred between the client and the server for EWT	{(AJAX data Response)}

Estimated Wait Time

Estimated Wait Time (EWT) is displayed in the ChannelSelector, Callback and ClickToCall Widgets. These Widgets use the getStats command to fetch EWT data from the GMS or GES server. These servers support multiple API versions and this document will explain how to configure the StatsService plugin to utilize version that you require.

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

Sample configuration:

```
_genesys.widgets.stats.ewt.apiVersion = <version value>
```

API Versions

v1

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

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

Default Example

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

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

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

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

v2

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

Example

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

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

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

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

v3

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

Example

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

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

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

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

Where to look for EWT data

When the getStats command is called, it fetches the EWT data from either GMS/GES server based on the configuration. This response data is included in the updated event in a standard format as shown below. In this data format, the **ewt** section will contain the virtual queue name and the estimated

wait time as a key value pair. The **response** section contains the original raw data from the server and may vary between each server API.

```
{
    ewt: {

        "VQ_GMS_Callback_Out":  9.999   // consolidated standardized EWT data for each
virtual queue.
        "VQ_GMS_Callback": 5.12
        ...
    },
    response: {                 // Original raw data from GMS.

        "VQ_GMS_Callback_Out": {
            "time": 1506021728,
            "wt": 0,
            "calls": 0,
            "wcalls": 0,
            "pos": 1,
            "wpos": 1,
            "aqt": 9.999,
            "ewt": 9.999,
            "hit": 0
        },
        "VQ_GMS_Callback": {
            ...
        }
    }
}
```

Toaster

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

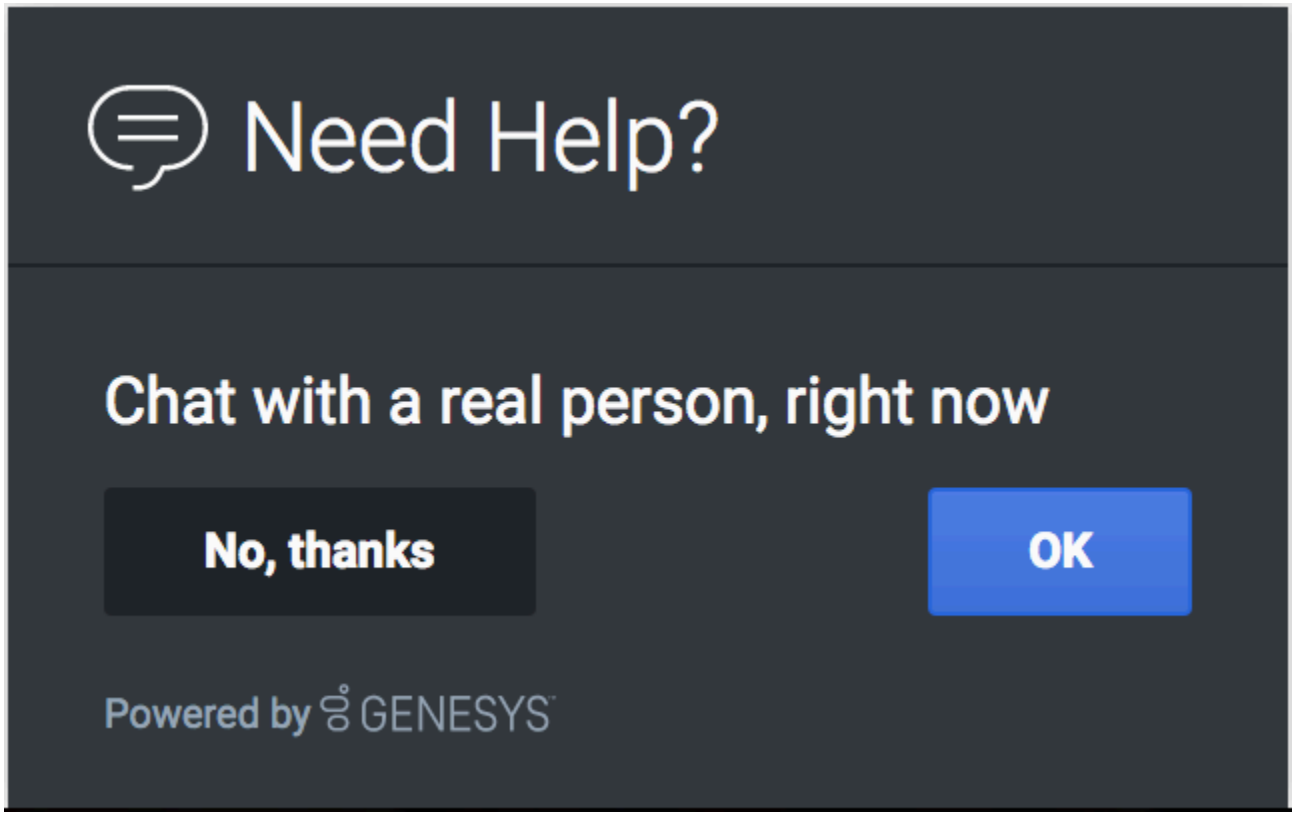
Customization

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	Accepted Values
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. **Important** The value `generic` places your content inside the common container style so the look and feel matches widgets. The value `custom` places your content inside a div container. It is then up to you to style your content.	generic, custom
title	string	Heading title to display on the toaster window.	n/a
body	string	Holds text value for Generic toaster type and html string template for Custom toaster type.	n/a
icon	string	The CSS class name for an icon.	n/a
controls	string	Show close and minimize controls on toaster window.	close, minimize, all
buttons	object	Define the type of buttons.	n/a
buttons.type	string	Shows two buttons on the toaster.	binary
buttons.primary	string	Text to be shown on primary button.	n/a
buttons.secondary	string	Text to be shown on secondary button.	n/a
immutable	boolean	When set to true, toaster cannot be closed by other plugins.	true, false

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
opened	Toaster plugin has appeared on the screen	n/a

WindowManager

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.

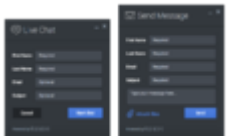
Customization

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)}