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.

Service Client API Reference

2/21/2026

Table of Contents

Get started

Namespaces

Search the table of all articles in this guide, listed in alphabetical order, to find the article you need.

Service Client API

Contents

Learn how to use the Service Client API to customize the way your web application integrates with Agent Workspace.

Important

Depending on your environment, you might need to contact your Genesys representative to complete the configuration described on this page.

Use the Service Client API to customize how your web application or website integrates with Agent Workspace. This JavaScript API is based on window.postMessage and provides methods your application can use to communicate cross domain with Agent Workspace while maintaining secured isolation.

Getting started

Here's an overview of the steps to access the API:

1. You have a web application that you've integrated in Agent Workspace. See Enabling integration of web applications in the agent interface.
2. Download the sample application from GitHub.
3. Copy the **wwe-service-client-api.js** file in the sample application to a location your web application can access.
4. Set configuration options related to security. See Security configuration.
5. Review Working with the API for more information about how to use the API.
6. Review the methods and types available in each namespace:
    - Agent Namespace
    - Configuration Namespace
    - Email Namespace
    - Interaction Namespace
    - Media Namespace
    - System Namespace
    - Voice Namespace
    - Outbound Namespace
    - Auth Namespace
    - Messenger Namespace

7. See Common actions with Service Client API for ideas about how to use the API.

Security configuration

The Service Client API involves two parties inside the agent's web browser: the service (the main web page) and the client (in an iframe on the same web page as the service). In order for the client web page to access the API, you need to set a few configuration options to work around web browser security restrictions for cross-origin requests and to enable request limits. You set these options on the **WWEWS Cluster** application only at the Application level; you can't set these options at the Agent or Agent Group level. Check out the Enabling the Service Client API topic in the *Workspace Web Edition Configuration Guide* for a full list of the options available to configure the API.

Origin

First, to work around web browser security restrictions set the service-client-api.accepted-web-content-origins option to the domain you want to be able to access to the API. For example, if you want to give access to a web page located at `http://my-web-server/path/page.html`, then you would set **service-client-api.accepted-web-content-origins** to `http://my-web-server`.

If you have several pages that need access to the API and they're located at different domains, you can also provide **service-client-api.accepted-web-content-origins** with a list. For example: `http://my-web-server, http://my-second-web-server, http://my-third-web-server`.

Finally, if you want to allow *any* page to access the API, just set **service-client-api.accepted-web-content-origins** to *.

You can also set the **service-client-api.accepted-web-content-origins** option to values that filter by API request, using any of the following keywords:

- agent.get
- agent.getState
- agent.getStateList
- agent.setState
- email.create
- interaction.deleteUserData
- interaction.getByInteractionId
- interaction.getInteractions
- interaction.selectCaseByCaseId
- interaction.setUserData
- interaction.singleStepTransfer(interactionId, targetQuery, userData, extensions, succeeded, failed)
- interaction.singleStepConference(interactionId, targetQuery, userData, extensions, succeeded, failed)
- interaction.consult(interactionId, targetQuery, userData, extensions, succeeded, failed)
- interaction.completeTransfer(consultInteractionId, succeeded, failed)

- interaction.completeConference(consultInteractionId, succeeded, failed)
- media.getMediaList
- media.setState
- voice.dial
- voice.dialEx(destination, userData, extensions, succeeded, failed)
- voice.pauseCallRecording
- voice.resumeCallRecording
- voice.startCallRecording
- voice.stopCallRecording

For example, you could set **service-client-api.accepted-web-content-origins** to `http://my-web-server0, http://my-web-server1 (*), http://my-web-server2 (agent.*, voice.dial), http://my-web-server3 (agent.*, interaction.*)`. In this example, everything is allowed for the `http://my-web-server0` and `http://my-web-server1`. For the `http://my-web-server2` domain, only the `agent.get`, `agent.getStateList`, `agent.setState`, `agent.getState` and `voice.dial` requests are allowed.

As seen in the example above, you can also filter by wildcards, using the asterisk in parenthesis. For example, `http://my-web-server1 (*)` or `http://my-web-server3 (agent.*, interaction.*)`.

Rate Limit

You can limit the maximum number of requests per minute on any Service Client API request by setting the service-client-api.rate-limit option. For example, setting the value to `50` would restrict the number of requests to 50 per minute. Set the value to `0` for unlimited requests.

If you want to limit the maximum number of requests per minute on a particular Service Client API request, use service-client-api.rate-limit..

Consider the following sample configuration:

```
service-client-api.rate-limit=0
service-client-api.rate-limit.voice.dial=4
service-client-api.rate-limit.email.create=2
```

In this example, there are no limits globally, but `voice.dial` requests are limited to 4 requests per minute and `email.create` requests are limited to 2 requests per minute.

Workspace calculates the limitation as a fixed interval of time, each minute (this is not calculated on a costly sliding window).

When the number of requests reaches the limit, Workspace ignores all further requests of the same type for a configurable period of time, known as the quarantine delay. In response, Workspace Web Edition sends a result with an explicit error message to the first request it receives after the limit is reached:

```
{
    "errorMessage": "The rate limit for the request 'voice.dial' has been reached.\nFurther
requests of the same type will be ignored for 30 seconds.",
    "request": "agent.getState"
}
```

To specify the global quarantine delay, set the service-client-api.rate-limit-quarantine-delay option. For example, setting the option to `60` means that Workspace Web Edition ignores requests for 60 seconds after the limit is reached. A value of `0` means that Workspace Web Edition ignores further requests forever, so use this value carefully.

Attached Data Access

Workspace offers two configuration options to limit the read or write access to the key/value pairs in user data:

- service-client-api.user-data.write-allowed specifies the list of keys in user data that can be written with the interaction.setUserData() or interaction.deleteUserData() functions.
- service-client-api.user-data.read-allowed specifies the list of keys in user data that can be read. This applies in the userData property of the interaction.deleteUserData() object returned by a function or an event.

For example, consider the following configuration:

```
service-client-api.user-data.write-allowed=Key1,Key3
service-client-api.user-data.read-allowed=Key1,Key2,Key3
```

This configuration lets you read the attached data with they keys Key1, Key2, and Key3, but only allows writes on keys Key1, and Key3.

Working with the API

After you've completed the setup and security steps, you're ready to start working with the Service Client API. The first thing you need to do is add a tag to your web application that points to the **wwe-service-client-api.js** file (remember, you stored it somewhere accessible in Step 3 above).

Now you can access the API through the **genesys.wwe.service** namespace. For example:

```
Hello world
```

Here's an example of how you could modify attached data:

```
genesys.wwe.service.interaction.setUserData(
    "1",
    {
        MyKEY1: "MyValue1",
        MyKEY2: "MyValue2"
    }
)
```

In the above example, the request is interaction.setUserData and the parameters are the `interactionId` of `1` and the `keyValues` of `MyKEY1` and `MyKEY2`. All methods provided in the Service Client API are asynchronous, so to get the successful or failed result, just add the matching callback:

```
genesys.wwe.service.interaction.setUserData(
    "1",
    {
        MyKEY1: "MyValue1",
        MyKEY2: "MyValue2"
    },
    function(result){
        console.debug("SUCCEEDED, result: " + JSON.stringify(result, null, '\t'));
    },
    function(result){
        console.debug("FAILED, result: " + JSON.stringify(result, null, '\t'));
    }
)
```

The global template for a service call is:

```
genesys.wwe.service..(<... function parameters ...>, [, []]);
```

The `done()` callback is called when a request is successfully sent without an error.

The `fail()` callback is called when a request generates an error or an exception.

The result of these functions is provided in a JSON object as a unique parameter.

Notifications

Warning

You must call `genesys.wwe.service.subscribe` only once.

You can use the following code to subscribe to **agent** and **interaction** notifications:

```
function eventHandler(message) {
    console.debug("Event: " + JSON.stringify(message, null, '\t'));
}

genesys.wwe.service.subscribe([ "agent", "interaction" ], eventHandler, context);
```

In the above example, `eventHandler` is the event handler function and `context` is an optional contextual object. Here's an example with an agent `STATE_CHANGED` to Ready:

```
{
    "event": "agent",
    "data": {
        "eventType": "STATE_CHANGED",
        "mediaState": "READY"
    }
}
```

Here's an example with an agent `STATE_CHANGED` to Not Ready with a reason:

```
{
    "event": "agent",
    "data": {
        "eventType": "STATE_CHANGED",
        "mediaState": "NOT_READY_ACTION_CODE",
        "reason": "Break",
        "reasonCode": "1511"
    }
}
```

Finally, here's an example with an `ATTACHED_DATA_CHANGED` event on a voice interaction:

```
{
        "event": "interaction",
        "data": {
                "eventType": "ATTACHED_DATA_CHANGED",
                "media": "voice",
                "interaction": {
                        "interactionId": "1",
                        "caseId": "4dda1ab6-aeab-4a33-f5d0-0153c9fdb43b",
                        "userData": {
                                "IWAttachedDataInformation": {
                                        "DispositionCode.Label": "DispositionCode",
                                        "Option.interaction.case-data.header-foreground-
color": "#FFFFFF",
                                        "CaseDataBusinessAttribute": "CaseData",
                                        "DispositionCode.Key": "ChooseDisposition",
                                        "Option.interaction.case-data.frame-color": "#17849D"
                                },
                                "IW_CaseUid": "4dda1ab6-aeab-4a33-f5d0-0153c9fdb43b",
                                "IW_BundleUid": "dfaca66c-4149-42a1-7244-337e949a12b5"
                        },
                        "parties": [
                                {
                                        "name": "5001"
                                }
                        ],
                        "callUuid": "4L6JGNEE9H7DT671FRPTKE6CQ000000G",
                        "state": "DIALING",
                        "previousState": "UNKNOWN",
                        "isConsultation": false,
                        "direction": "OUT",
                        "callType": "Internal",
                        "dnis": "5001",
                        "isMainCaseInteraction": true
                }
        }
}
```

Event Type references

The system eventType field can be one of the following:

eventType	Description
CUSTOM_TOAST_BUTTON_CLICK	Uses the following parameters: • **customToastId**: The identifier of the toast where the button has been clicked. The identifier is returned by the popupToast method. • **buttonIndex**: The index of the clicked button. The index starts by 0.
REALTIME_CONNECTION	Uses the following parameters: • **state**: The attribute can take any of the following values: • DISCONNECTED - The real-time connection with the Genesys Web Services server is disconnected. • RECONNECTED - The real-time connection with the Genesys Web Services server is established after a disconnection. • DOWN - The real-time connection with the Genesys Web Services server is down for more than one minute due to server inactivity. In this situation, we can consider the session as *Down*.

The interaction eventType field can be one of the following:

eventType	Description
Common events to all interaction types	
UNKNOWN	An unknown event occurs.
ADDED	The interaction has been added in the list of interactions.
REMOVED	The interaction has been removed from the list of interactions.
ATTACHED_DATA_CHANGED	The attached data have changed in the interaction.
CASE_OR_BUNDLE_ID_CHANGED	The case or the bundle identifier of this interaction has changed.
CASE_ID_CHANGED	The case identifier of this interaction has changed.
NEW_MESSAGE	This event represents a new message.
ERROR	An error occurs in the interaction.
CONTACT_CHANGED	A contact associated with the interaction is fully or partially modified.
Voice events	
CALL_RECORDING_STATE_CHANGED	The call recording state changed.

eventType	Description
DIALING	The outbound call starts ringing.
ESTABLISHED	The call has been established.
HELD	The call has been held.
PARTY_CHANGED	The list of party has been changed in the interaction.
RELEASED	The call has been released.
RINGING	The inbound call starts ringing.
OpenMedia events	
ACCEPTED	The open media interaction is accepted.
COMPLETED	The open media interaction has been completed (Mark as done).
COMPOSING	The open media interaction is in composing mode.
CREATED	The open media interaction has been created.
INSERT_STANDARD_RESPONSE	A standard response has been inserted in the interaction.
INVITED	The open media interaction is an invitation.
INVITED_CONFERENCE	The open media interaction receive a conference invitation.
IN_QUEUE_FAILED	The place in queue has failed.
IN_WORKBIN	The interaction has been placed in the work-bin.
IN_WORKBIN_FAILED	The place in work-bin has failed.
LEFT_CONFERENCE	The open media interaction has left the conference.
PULLED	The open media interaction has been pulled from a work-bin.
PULL_FAILED	The pull from the queue has failed.
PULL_WORKBIN_FAILED	The pull from the work-bin has failed.
REVOKED	The open media interaction has been revoked.
TRANSFER_COMPLETED	The open media interaction has been transferred and the transfer has been completed.
Chat events (inherit from OpenMedia events)	
CANCELED	The interaction is already accepted in another chat session.
ENDED	The chat has been ended.
JOIN_FAILED	The connection with the chat server failed.
JOIN_PENDING	The interaction is trying to join the chat session.
Outbound email events (inherit from OpenMedia events)	
CANCELLED	The outbound email has been cancelled.
SENT	The outbound email has been sent.

Outbound events

The **Outbound preview events** table lists the SCAPI event details for Pull Preview, Push Preview and Direct Push Preview records.

Outbound preview events

Mode	UI Event	Event Type	State	Call Type	Capabilities
Pull Preview	Preview record received	ADDED	PREVIEWING	OUTBOUND_PREVIEW	CALL, REJECT_RECORD, CANCEL_RECORD
		PREVIEWING	PREVIEWING	OUTBOUND_PREVIEW	CALL, REJECT_RECORD, CANCEL_RECORD
	Make call from preview	ADDED	DIALING	OUTBOUND	HANGUP
		DIALING	DIALING	OUTBOUND	HANGUP
		REMOVED	IDLE	OUTBOUND_PREVIEW	-
	Release and mark done	RELEASED	IDLE	OUTBOUND	MARK_DONE
		MARKDONE_APPLY	IDLE	OUTBOUND	MARK_DONE
		REMOVED	IDLE	OUTBOUND	-
	Reject record	STATE_CHANGE	REJECTED	OUTBOUND_PREVIEW	MARK_DONE
	Cancel record	STATE_CHANGE	CANCELED	OUTBOUND_PREVIEW	MARK_DONE
Regular Push Preview	Record received	ADDED	INVITED	OUTBOUND_PUSH_PREVIEW	ACCEPT, REJECT
		INVITED	INVITED	OUTBOUND_PUSH_PREVIEW	ACCEPT, REJECT
	Accepted	PREVIEWING	PREVIEWING	OUTBOUND_PUSH_PREVIEW	CALL, REJECT_RECORD, CANCEL_RECORD
	Rejected	REMOVED	REJECTED	OUTBOUND_PUSH_PREVIEW	-
	Make call	ADDED	DIALING	OUTBOUND	HANGUP
		DIALING	DIALING	OUTBOUND	HANGUP
		ESTABLISHED	TALKING	OUTBOUND	HANGUP, HOLD
	Release and mark done	RELEASED	IDLE	OUTBOUND	MARK_DONE
		MARKDONE_APPLY	IDLE	OUTBOUND	MARK_DONE
		REMOVED	IDLE	OUTBOUND_PUSH_PREVIEW	MARK_DONE
		REMOVED	IDLE	OUTBOUND	-
	Reject record	STATE_CHANGE	REJECTED	OUTBOUND_PUSH_PREVIEW	MARK_DONE
	Cancel record	STATE_CHANGE	CANCELED	OUTBOUND_PUSH_PREVIEW	MARK_DONE
Direct Push Preview	Record received	ADDED	INVITED	OUTBOUND_PREVIEW	ACCEPT, REJECT
		INVITED	INVITED	OUTBOUND_PREVIEW	ACCEPT, REJECT
	Accepted	PREVIEWING	PREVIEWING	OUTBOUND_PREVIEW	CALL,

Mode	UI Event	Event Type	State	Call Type	Capabilities
					REJECT_RECORD, CANCEL_RECORD
	Rejected	REMOVED	REJECTED	OUTBOUND_PREVIEW	
	Make call	ADDED	DIALING	OUTBOUND	HANGUP
		DIALING	DIALING	OUTBOUND	HANGUP
		ESTABLISHED	TALKING	OUTBOUND	HANGUP
		REMOVED	IDLE	OUTBOUND_PREVIEW	
	Release and mark done	RELEASED	IDLE	OUTBOUND	MARK_DONE
		MARKDONE_APPLY	IDLE	OUTBOUND	MARK_DONE
		REMOVED	IDLE	OUTBOUND	-
	Reject record	STATE_CHANGE	REJECTED	OUTBOUND_PREVIEW	MARK_DONE
	Cancel record	STATE_CHANGE	CANCELED	OUTBOUND_PREVIEW	MARK_DONE

The **Outbound campaign events** table lists the possible events for outbound campaigns.

Outbound campaign events

EventType	Trigger	Example

CampaignLoaded	When an outbound campaign is loaded.	`{`
		`    "event": "outbound",`
		`    "data": {`
		`        "eventType": "CampaignLoaded",`
		`        "campaign": "Offer of the Month"`
		`    },`
		`    "userAgent": "WWE Server",`
		`    "protocolVersion": 2`
		`}`

CampaignUnloaded	When an outbound campaign is unloaded.	`{`
		`    "event": "outbound",`
		`    "data": {`
		`        "eventType": "CampaignUnloaded",`
		`        "campaign": "Offer of the Month"`
		`    },`
		`    "userAgent": "WWE Server",`
		`    "protocolVersion": 2`
		`}`

CampaignStarted	When an outbound campaign starts.	This event also has a "mode" property that describes the mode in which the campaign started. `{`

EventType	Trigger	Example

		"event": "outbound",
		"data": {
		"eventType": "CampaignStarted",
		"campaign": "Offer of the Month",
		"mode": "Predictive GVP"
		},
		"userAgent": "WWE Server",
		"protocolVersion": 2
		}

CampaignStopped	When an outbound campaign stops.	{
		"event": "outbound",
		"data": {
		"eventType": "CampaignStopped",
		"campaign": "Offer of the Month"
		},
		"userAgent": "WWE Server",
		"protocolVersion": 2
		}

Chain of records events

The RECORDS_RETRIEVED event is triggered on an outbound interaction when all of the records in the interaction's chain of records have been retrieved.

Sample response

```
{
    "event": "interaction",
    "data": {
        "eventType": "RECORDS_RETRIEVED",
        "interaction": {
            "interactionId": "1",
            "caseId": "a26f59d2-2979-43c5-5c1d-b0757f9ab077",
            "parentInteractionId": null,
            "chainedRecords": [
                {
                    Custom_Character: "c"
                    Custom_Datetime: "2021-03-17 14:42:39"
                    Custom_Float: "16.64"
                    Custom_Integer: 0
                    Custom_String_with_default: "Hi there!           "
                    Custom_VarChar: ""
                    GSW_AGENT_ID: "+33298025000"
                    GSW_APPLICATION_ID: 139
                    GSW_ATTEMPTS: 0
                    GSW_CALLING_LIST: "Calling List Custom"
                    GSW_CALLING_LIST_DBID: 101
                    GSW_CALL_ATTEMPT_GUID: "003DC7H6HG84DBRT1KMIF1TAES000031"
                    GSW_CALL_RESULT: 28
                    GSW_CAMPAIGN_GROUP_DBID: 101
                    GSW_CAMPAIGN_GROUP_DESCRIPTION: ""
                    GSW_CAMPAIGN_GROUP_NAME: "Outbound Campaign Custom@Agent Group Outbound"
                    GSW_CAMPAIGN_NAME: "Outbound Campaign Custom"
                    GSW_CHAIN_ID: 3
                    GSW_CONTACT_MEDIA_TYPE: "voice"
                    GSW_FROM: 0
                    GSW_PHONE: "+33647005"
                    GSW_PHONE_TYPE: 1
                    GSW_RECORD_HANDLE: 283
                    GSW_REFERENCE_ID: 3
                    GSW_SWITCH_DBID: 101
                    GSW_TZ_NAME: "ACT"
                    GSW_TZ_OFFSET: 34200
                    GSW_UNTIL: 86399
                    GSW_USER_EVENT: "PreviewRecord"
                    IW_BundleUid: "27458420-0348-4345-c693-45bd95b5c81f"
                    IW_CaseUid: "a26f59d2-2979-43c5-5c1d-b0757f9ab077"
                    InteractionSubtype: "OutboundNew"
                    InteractionType: "Outbound"
                    WWE_OUTBOUND_CAMP_TYPE: "PreviewRecord"
                },
                {
                    Custom_Character: "c"
                    Custom_Datetime: "2021-03-17 14:42:32"
                    Custom_Float: "51.69"
                    Custom_Integer: 0
                    Custom_String_with_default: "Hello General Kenobi"
                    Custom_VarChar: ""
                    GSW_AGENT_ID: "+33298025000"
                    GSW_APPLICATION_ID: 139
                    GSW_ATTEMPTS: 0
                    GSW_CALLING_LIST: "Calling List Custom"
                    GSW_CALLING_LIST_DBID: 101
                    GSW_CALL_ATTEMPT_GUID: "003DC7H6HG84DBRT1KMIF1TAES000031"
                    GSW_CALL_RESULT: 28
                    GSW_CAMPAIGN_GROUP_DBID: 101
                    GSW_CAMPAIGN_GROUP_DESCRIPTION: ""
                    GSW_CAMPAIGN_GROUP_NAME: "Outbound Campaign Custom@Agent Group Outbound"
                    GSW_CAMPAIGN_NAME: "Outbound Campaign Custom"
                    GSW_CHAIN_ID: 3
                    GSW_CONTACT_MEDIA_TYPE: "voice"
                    GSW_FROM: 0
                    GSW_PHONE: "+33647004"
                    GSW_PHONE_TYPE: 1
                    GSW_RECORD_HANDLE: 284
                    GSW_REFERENCE_ID: 4
                    GSW_SWITCH_DBID: 101
                    GSW_TZ_NAME: "ACT"
                    GSW_TZ_OFFSET: 34200
                    GSW_UNTIL: 86399
                    GSW_USER_EVENT: "ChainedRecord"
                    InteractionSubtype: "OutboundNew"
                    InteractionType: "Outbound"
                }
            ],
            "userData": {
                "GSW_PHONE": "+33647005",
                "GSW_PHONE_TYPE": "1",
                "Custom_Character": "c",
                "Custom_Datetime": "2021-03-17 14:42:39",
                "Custom_Float": "16.64",
                "Custom_Integer": "0",
                "Custom_String_with_default": "Hi there!           ",
                "Custom_VarChar": "",
                "GSW_FROM": "0",
                "GSW_UNTIL": "86399",
                "GSW_TZ_OFFSET": "34200",
                "GSW_CALLING_LIST": "Calling List Custom",
                "GSW_CAMPAIGN_NAME": "Outbound Campaign Custom",
                "InteractionType": "Outbound",
                "InteractionSubtype": "OutboundNew",
                "GSW_RECORD_HANDLE": "283",
                "GSW_APPLICATION_ID": "139",
                "GSW_CAMPAIGN_GROUP_DBID": "101",
                "GSW_CALLING_LIST_DBID": "101",
                "GSW_CAMPAIGN_GROUP_NAME": "Outbound Campaign Custom@Agent Group Outbound",
                "GSW_CAMPAIGN_GROUP_DESCRIPTION": "",
                "GSW_CHAIN_ID": "3",
                "GSW_ATTEMPTS": "0",
                "GSW_CALL_RESULT": "28",
                "GSW_TZ_NAME": "ACT",
                "GSW_CALL_ATTEMPT_GUID": "003DC7H6HG84DBRT1KMIF1TAES000031",
                "GSW_CONTACT_MEDIA_TYPE": "voice",
                "GSW_REFERENCE_ID": "3",
                "GSW_SWITCH_DBID": "101",
                "GSW_USER_EVENT": "PreviewRecord",
                "GSW_AGENT_ID": "+33298025000",
                "WWE_OUTBOUND_CAMP_TYPE": "PreviewRecord",
                "IW_BundleUid": "27458420-0348-4345-c693-45bd95b5c81f",
                "IW_CaseUid": "a26f59d2-2979-43c5-5c1d-b0757f9ab077"
            },
            "state": "PREVIEWING",
            "previousState": "UNKNOWN",
            "capabilities": [
                "CALL",
                "REJECT_RECORD",
                "CANCEL_RECORD"
            ],
            "parties": [
                {
                    "name": "+33647005"
                }
            ],
            "startDate": null,
            "endDate": null,
            "callType": "OUTBOUND_PREVIEW",
            "isMainCaseInteraction": true,
            "isCaseSelected": true,
            "isCaseExpanded": false
        }
    },
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

Common actions with Service Client API

The following sections show some common actions you can perform with Service Client API:

Controlling call recording from a third-party application

Review the following methods for details about call recording control:

- pauseCallRecording
- resumeCallRecording
- startCallRecording
- stopCallRecording

The call recording state is stored in the `recordingState` attribute on the interaction.Interaction object.

Embedding multiple third-party applications in Agent Workspace

You can configure Agent Workspace to include more than one third-party web application, displayed as either a tab, a popup window, in the background at the interaction level, or hidden. Configure the following options:

- Set the interaction.web-content option to a list of option section names that correspond to web extension views.
- Make sure that the service-client-api.accepted-web-content-origins option references all the websites that should use the Service Client API.

Updating attached data from a third-party application

Review the following methods for details about updating attached data:

- deleteUserData
- getByInteractionId
- getInteractions
- setUserData

The user data is stored in the `userData` attribute on the interaction.Interaction object.

You should also set the options related to user data in the Service Client section of Agent Setup or configure the service-client-api.user-data.read-allowed and service-client-api.user-data.write-allowed options.

Enabling click-to-dial from a third-party application

If you configure Agent Workspace to display your web application in a new tab in the Agent Workspace user interface, then the service API only gives access to the dial operation.

Enabling Service Client API to invoke toast in Agent Workspace

Review the following methods for details about enabling and updating toast:

- system.popupToast
- system.updateToast
- system.closeToast

Controlling case selection from a third-party application

Review the following method for details about case selecting control:

- selectCaseByCaseId

The case selection state is stored in the *isCaseSelected* attribute and the *isCaseExpanded* attribute on the **interaction.Interaction** object.

Supporting multiple browser tabs

Service Client API supports multiple browser tabs in a session. The API uses the concept of a leader tab and following tab or tabs. When multiple tabs are open, certain actions (typically automatic) are performed only by the leader tab, such as auto-answer for chat, email, and voice interactions, and contact management in Universal Contact Server. The API also tracks which tab was the last active because some actions are performed only by this tab, such as sounds, toasts, and supervisor-forced log out.

The state of a given browser tab is determined by an internal election process, which can be triggered when an agent closes a leader tab. The state is exposed through the **data.frameState** property on system events. The **frameState** property has three possible values:

- LEADING: The election happened and this tab is the leader.
- FOLLOWING: The election happened and this tab is a follower.
- NEGOTIATING: The election is in progress and no tab is a leader or follower until the election is finished.

You can subscribe to system events as follows:

```
function eventHandler(message) {
  switch (message.event) {
    case 'system':
      log('Received system event: ', JSON.stringify(message, null, '\t'));
      break;
    default:
      break;
  }
}

genesys.wwe.service.subscribe(['system'], eventHandler, this);
```

When an election is triggered, you should see these types of system events:

```
Received system event:
{
    "event": "system",
    "data": {
        "frameState": "LEADING"
    },
    "userAgent": "WWE Server",
    "protocolVersion": 2
}

Received system event:
{
    "event": "system",
    "data": {
        "frameState": "NEGOTIATING"
    },
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

Service Client API provides some helper functions through the System namespace to determine the state of a tab:

- isFrameLeading
- isFrameFollowing
- isFrameNegotiating
- isFrameLeadingOrNegotiating
- isLastActiveFrame

Service Client API updates the attached data for an interaction in the leader tab with a new **caseId** on eventType CASE_ID_CHANGED.

```
{
    "event": "interaction",
    "data": {
        "eventType": "CASE_ID_CHANGED",
        "caseId": "e6470563-af78-4942-657d-976a25dd9de3",
        "previousCaseId": "5f7e5f3a-fb6e-43f3-c404-eaee21d64ef1"
    },
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

Agent namespace

Contents

Learn about the Agent namespace methods and type definitions in the Service Client API.

Methods

The Agent namespace includes the following methods:

- get
- getState
- getStateList
- setState

get

<table>
<tr><th>Signature</th><td>get(<i>succeeded, failed</i>) → {agent.Agent}</td></tr>
<tr><th>Description</th><td>Gets the agent's attributes.</td></tr>
<tr><th>Parameters</th><td><table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table></td></tr>
<tr><th>Returns</th><td>agent.Agent</td></tr>
</table>

getState

<table>
<tr><th>Signature</th><td>getState(<i>succeeded, failed</i>) → {media.State}</td></tr>
<tr><th>Description</th><td>Gets the agent's state.</td></tr>
<tr><th>Parameters</th><td><table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Signature</th><th>getState(<em>succeeded, failed</em>) → {media.State}</th></tr>
<tr><td></td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>the operation fails.</td></tr></table></td></tr>
<tr><td><strong>Returns</strong></td><td>media.State</td></tr>
</table>

getStateList

<table>
<tr><th>Signature</th><th>getStateList(<em>succeeded, failed</em>) → {Array.}</th></tr>
<tr><td><strong>Description</strong></td><td>Gets the list of possible agent states.</td></tr>
<tr><td><strong>Parameters</strong></td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table></td></tr>
<tr><td><strong>Returns</strong></td><td>Array.</td></tr>
</table>

setState

<table>
<tr><th>Signature</th><th>setState(<em>stateOperationName, succeeded, failed</em>)</th></tr>
<tr><td><strong>Description</strong></td><td>Sets the agent's state.</td></tr>
<tr><td><strong>Parameters</strong></td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>stateOperationName</td><td>string</td><td>An <code>operationName</code> from the agent states list. See State.</td></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table></td></tr>
</table>

Type definitions

The agent namespace includes the following object types:

- Agent

Agent

<table>
<tr><th>Description</th><td>Represents the JSON structure of the agent.</td></tr>
<tr><th>Type</th><td>Object</td></tr>
<tr><th>Properties</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>employeeId</td><td>string</td><td>The agent's unique identifier used for routing purposes.</td></tr>
<tr><td>firstname</td><td>string</td><td>The agent's first name.</td></tr>
<tr><td>lastname</td><td>string</td><td>The agent's last name.</td></tr>
<tr><td>username</td><td>string</td><td>The agent's username. This is a global unique ID.</td></tr>
</table>
</td></tr>
</table>

Configuration namespace

Contents

Learn about the Configuration namespace methods and type definitions in the Service Client API.

> **Important**
> Depending on your environment, you might need to contact your Genesys representative to complete the configuration described on this page.

Methods

The Configuration namespace includes the following methods:

- getOption
- getContextualOption

getOption

<table>
<tr><td><b>Signature</b></td><td>getOption(<i>options</i>, succeeded, failed) → {Array.
}</td></tr>
<tr><td><b>Description</b></td><td>Get configuration options and values for a specific option name or a subset of options from the <b>[interaction-workspace]</b> section or a custom section.</td></tr>
<tr><td><b>Parameters</b></td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>options</td><td>string</td><td>An array of configuration options or sections to return. Unless otherwise specified, the API returns options from the <b>[interaction-workspace]</b> section by default. You can specify</td></tr></table></td></tr>
</table>

<table>
<tr><th>Signature</th><th>getOption(<i>options</i>, succeeded, failed) → {Array.
}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>any of the following:
• A single option: <code>genesys.wwe.service.configuration.getOption('voice.auto-answer', succeeded, failed)</code>
• A single option in a specific section: <code>genesys.wwe.service.configuration.getOption('CustomSection/option.custom.customer.code', succeeded, failed)</code>
• Multiple options: <code>genesys.wwe.service.configuration.getOption(['voice.auto-answer', 'privilege.email.can-mark-done'], succeeded, failed)</code>
• Multiple</td></tr>
</table>
</td></tr>
</table>

Signature	getOption(*options*, succeeded, failed) → {Array. }

Name	Type	Description
		options in different sections: `genesys.wwe.service.configuration.getOption(['privilege.*', 'CustomSection/option.custom.customer.code'], succeeded, failed)` You can use an asterisk '*' as a wildcard, but only at the end of each word. For example: • voice.* • voice.auto* • sipendpoint.* • CustomAPI/test.* You cannot use an asterisk at the start of an option or section. For example, the following values are not allowed: • *.mark-done • *.auto
succeeded	function	A function called when the operation succeeds.
failed	function	A function called when

<table>
<tr><th>Signature</th><th>getOption(*options*, succeeded, failed) → {Array.
}</th></tr>
<tr><td></td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>the operation fails.</td></tr></table></td></tr>
<tr><td>**Returns**</td><td>Array.</td></tr>
</table>

getContextualOption

<table>
<tr><th>Signature</th><th>getContextualOption(*options*, *interactionId*, succeeded, failed) → {Array.
}</th></tr>
<tr><td>**Description**</td><td>Get configuration options and values in the context of this interaction when they are overridden by a routing strategy. If the interaction is not found or this parameter is missing, the API throws an exception. Note: The getContextualOption method can be applied to any option, even if the option doesn’t support overriding options with a routing strategy in Workspace Web Edition. Only Workspace Web Edition options with the following text in their descriptions can be overridden by a routing strategy: "This option can be overridden by a routing strategy as described in this Configuration Guide." Using the getContextualOption method doesn’t interfere with how Workspace Web Edition handles options.</td></tr>
<tr><td>**Parameters**</td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>options</td><td>string</td><td>An option or array of options and their values. Unless otherwise specified, the API returns options from the **[interaction-workspace]** section by default. You can specify any of the following:
• A single option by name:</td></tr></table></td></tr>
</table>

<table>
<tr><th>Signature</th><th>getContextualOption(<i>options</i>, <i>interactionId</i>, succeeded, failed) → {Array.
}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td><code>genesys.wwe.service.configuration.getOption('voice.auto-answer', succeeded, failed)</code>
• A subset of options defined by '*':
• A subset of options in a specific section: <code>genesys.wwe.service.configuration.getOption('interaction-workspace/interaction.case-data.*', succeeded, failed)</code>
You can use an asterisk '*' as a wildcard, but only at the end of each word. For example:
• voice.*
• voice.auto*
• sipendpoint.*
• CustomAPI/test.*
You cannot use an asterisk at the</td></tr>
</table>
</td></tr>
</table>

Signature	getContextualOption(*options*, *interactionId*, succeeded, failed) → {Array. }
	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>start of an option or section. For example, the following values are not allowed:<ul><li>*.mark-done</li><li>*.auto</li></ul></td></tr><tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table>
Returns	Array.

Type definitions

The Configuration namespace includes the following object types:

- Section

Section

Description	Represents the JSON structure of a configuration section. Each section includes a list of key/value pairs for the matching option(s).
Type	Object
Properties	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>name</td><td>string</td><td>The name of the configuration option.</td></tr></table>

<table>
<tr><th>Description</th><td>Represents the JSON structure of a configuration section. Each section includes a list of key/value pairs for the matching option(s).</td></tr>
<tr><td></td><td><table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>value</td><td>string or array of strings</td><td>The value of the configuration option.</td></tr>
</table></td></tr>
</table>

Examples

Consider the following scenario:

1. You set `interaction-workspace/interaction.case-data.frame-color"="#FFBA00"`.
2. You also set the override option key: `"interaction-workspace/interaction.override-option-key"= "IW_OverrideOptions"`.
3. An interaction arrives with the attached data `"IW_OverrideOptions"="CaseDataColor"`.
4. The transaction object "CaseDataColor" annex has the option `"interaction-workspace/ interaction.case-data.frame-color"="#FF000088"`.

Here's how this scenario would look for each of the Configuration namespace methods:

getContextualOption()

Use getContextualOption() to get the option:

```
genesys.wwe.service.configuration.getContextualOption("interaction.case-data.frame-color",
"1", succeeded, failed)
```

You receive this response:

```
{
    "request": "configuration.getContextualOption",
    "data": {
        "interaction-workspace": {
            "interaction.case-data.frame-color": "#FF008000"
        }
    },
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

If the interaction isn't found, the failed callback receives this response:

```
{
    "request": "configuration.getContextualOption",
    "errorMessage": "Error: Interaction not found.",
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

getOption()

Use getOption() to get the option::

```
genesys.wwe.service.configuration.getOption("interaction.case-data.frame-color", succeeded,
failed)
```

The response includes the original default value of the option instead of the overridden value:

```
{
    "request": "configuration.getOption",
    "data": {
        "interaction-workspace": {
            "interaction.case-data.frame-color": "#FFBA00"
        }
    },
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

Email namespace

Contents

Learn about the Email namespace methods in the Service Client API.

Methods

The Email namespace includes the following methods:

- create

create

<table>
<tr><th>Signature</th><td>create(*destination*, *userData, succeeded, failed*)</td></tr>
<tr><th>Description</th><td>Creates a new empty email.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>destination</td><td>string</td><td></td><td>The destination address for the email.</td></tr>
<tr><td>userData</td><td>object</td><td></td><td>The attached user data key/value object that is updated with each interaction event.</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

Interaction namespace

Contents

Learn about the Interaction namespace methods and type definitions in the Service Client API.

> **Important**
>
> Depending on your environment, you might need to contact your Genesys representative to complete the configuration described on this page.

Methods

The Interaction namespace includes the following methods:

- completeConference
- completeTransfer
- consult
- deleteUserData
- getByInteractionId
- getInteractions
- selectCaseByCaseId
- setUserData
- markdone
- blockMarkdone
- singleStepConference
- singleStepTransfer
- unblockMarkdone
- accept
- reject

completeConference

Signature	completeConference(*consultInteractionId, succeeded, failed*)
Description	Completes a conference.

<table>
<tr><th>Signature</th><th>completeConference(<i>consultInteractionId, succeeded, failed</i>)</th></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>consultInteractionId</td><td>string</td><td>The unique identifier for the consultation interaction.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

completeTransfer

<table>
<tr><th>Signature</th><th>completeTransfer(<i>consultInteractionId, succeeded, failed</i>)</th></tr>
<tr><td><b>Description</b></td><td>Completes a transfer.</td></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>consultInteractionId</td><td>string</td><td>The unique identifier for the consultation interaction.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

consult

<table>
<tr><th>Signature</th><th>consult(<i>interactionId, targetQuery, userData, extensions, succeeded, failed</i>)</th></tr>
<tr><td><b>Description</b></td><td>Make a consultation interaction.</td></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>consult(<i>interactionId, targetQuery, userData, extensions, succeeded, failed</i>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>identifier for the interaction.</td></tr>
<tr><td>targetQuery</td><td>object or string</td><td>The destination target object, or a character string (for example, phone number).
<ul>
<li>If targetQuery is a character string, the Service Client API creates the operation that uses a target of type <b>CustomContact</b> with a destination set to this value.</li>
<li>If targetQuery is a JSON object, specify the following sub-parameters:
<ul>
<li><b>target (string)</b>: The target type. The possible values are: "AGENT", "AGENT_GROUP",</li>
</ul>
</li>
</ul>
</td></tr>
</table>
</td></tr>
</table>

Signature	consult(*interactionId, targetQuery, userData, extensions, succeeded, failed*)
	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>"SKILL", "INTERACTION_QUEUE", "ROUTING_POINT", and "CUSTOM_CONTACT". • **destination (string)**: The destination. The supported values are: the employeeId of an agent, the name of an AgentGroup, the name of a Skill, the name of an InteractionQueue, the name of a RoutingPoint, and a phone number for CustomContact. • **[media] (string)**: An optional media used to make the consultation. If not specified,</td></tr></table>

Signature	consult(*interactionId, targetQuery, userData, extensions, succeeded, failed*)
	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>uses the same media as the specified interaction. For example, if the interaction has a "chat" media, and you want to make a voice consultation, you must specify "voice" here.</td></tr><tr><td>userData</td><td>object</td><td>The attached user data key/value object. Set an undefined or empty JSON object if you don't want to set any user data.</td></tr><tr><td>extensions</td><td>object</td><td>The extensions key/value object. Set an undefined or empty JSON object if you don't want to set any extensions. This is not applicable for the chat media.</td></tr></table>

<table>
<tr><th>Signature</th><th>consult(*interactionId, targetQuery, userData, extensions, succeeded, failed*)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

deleteUserData

<table>
<tr><th>Signature</th><th>deleteUserData(*interactionId*, *key*, succeeded, failed)</th></tr>
<tr><td>Description</td><td>Deletes the user data attached to the interaction. The List of User Data Write Allowed setting in Agent Setup or the service-client-api.user-data.write-allowed configuration option might restrict the allowed key/value pairs.</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr>
<tr><td>key</td><td>string</td><td>The key to delete from the attached data.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

getByInteractionId

Signature	getByInteractionId(*interactionId*, succeeded, failed) → {interaction.Interaction}
Description	Gets an interaction by its unique identifier.

<table>
<tr><th>Signature</th><th>getByInteractionId(<i>interactionId</i>, succeeded, failed) → {interaction.Interaction}</th></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
<tr><td><b>Returns</b></td><td>interaction.Interaction or null if the interaction doesn't exist.</td></tr>
</table>

getInteractions

<table>
<tr><th>Signature</th><th>getInteractions(succeeded, failed) → {Array.}</th></tr>
<tr><td><b>Description</b></td><td>Gets all the interactions.</td></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
<tr><td><b>Returns</b></td><td>Array.</td></tr>
</table>

selectCaseByCaseId

<table>
<tr><th>Signature</th><th>genesys.wwe.service.interaction.selectCaseByCaseId(caseId, succeeded, failed)</th></tr>
<tr><td><b>Description</b></td><td>Select the case in the UI by case identifier. If you subscribe to the "interaction" events (genesys.wwe.service.subscribe(["interaction"], eventHandler, this);), you will receive the following event:
<pre>
Received interaction event: {
        "event": "interaction",
</pre>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>genesys.wwe.service.interaction.selectCaseByCaseId(caseId, succeeded, failed)</th></tr>
<tr><td></td><td>

```
        "data": {
                "eventType": 
"CASE_COLLAPSED",
                "selectedCaseId": "4401820b-
c4e6-4994-69c2-6ae7fdbc4905"
        },
        "userAgent": "WWE Server",
        "protocolVersion": 2
}
Received interaction event: {
        "event": "interaction",
        "data": {
                "eventType": 
"CASE_EXPANDED",
                "selectedCaseId": "4401820b-
c4e6-4994-69c2-6ae7fdbc4905"
        },
        "userAgent": "WWE Server",
        "protocolVersion": 2
}
Received interaction event: {
        "event": "interaction",
        "data": {
                "eventType": 
"CASE_SELECTED",
                "selectedCaseId": 
"d4187b87-9fe1-4db8-0515-6a91e666e22d"
        },
        "userAgent": "WWE Server",
        "protocolVersion": 2
}
```

</td></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>caseId</td><td>string</td><td>The unique identifier for the case.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

setUserData

Signature	setUserData(*interactionId*, *keyValues*, succeeded, failed)
Description	Sets the user data on the live interaction (for voice, this means the interaction is not in the IDLE state).

<table>
<tr><th>Signature</th><td>setUserData(*interactionId*, *keyValues*, succeeded, failed)</td></tr>
<tr><td></td><td>This request overwrites any existing keys on the user data. The List of User Data Write Allowed setting in Agent Setup or the service-client-api.user-data.write-allowed configuration option might restrict the allowed key/value pairs.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr>
<tr><td>keyValues</td><td>object</td><td>The key value pairs to set on the user data.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

markdone

<table>
<tr><th>Signature</th><td>markdone(*interactionId*, succeeded, failed)</td></tr>
<tr><th>Description</th><td>Mark done the selected interaction.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

Outbound interactions

The `markdone` operation can be used for outbound interactions such as pull preview, push preview, and direct push preview, but there are some details you need to know:

- Pull preview - Mark done is similar to doing a 'Done and Stop' action, where the next preview record is not fetched.
- Push preview - No special behaviour.
- Direct push preview - Mark done is similar to doing a 'Done and Stop' action, where it triggers a notification to Outbound Contact Server to stop sending direct push preview records.

blockMarkdone

<table>
<tr><th>Signature</th><td>blockMarkdone(<i>interactionId</i>, <i>warningMessage</i>, succeeded, failed)</td></tr>
<tr><th>Description</th><td>Block the mark done operation on the selected interaction. The "markdone" event must be subscribed to receive the event which informs that there is a delay in blocking the markdone operation with this method.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique interaction identifier of the interaction to prevent the mark done operation.</td></tr>
<tr><td>warningMessage</td><td>string</td><td>The warning message.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

singleStepConference

<table>
<tr><th>Signature</th><td>singleStepConference(<i>interactionId, targetQuery, userData, extensions, succeeded, failed</i>)</td></tr>
<tr><th>Description</th><td>Make a single step conference.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr>
<tr><td>targetQuery</td><td>object or string</td><td>The destination</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>singleStepConference(<em>interactionId, targetQuery, userData, extensions, succeeded, failed</em>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>target object, or a character string (for example, phone number).
- If targetQuery is a character string, the Service Client API creates the operation that uses a target of type <strong>CustomContact</strong> with a destination set to this value.
- If targetQuery is a JSON object, specify the following sub-parameters:
  - <strong>type (string)</strong>: The target type. The possible values are: "AGENT", "AGENT_GROUP", "SKILL", "INTERACTION_QUEUE", "ROUTING_POINT", and "CUSTOM_CONTACT".</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>singleStepConference(<i>interactionId, targetQuery, userData, extensions, succeeded, failed</i>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>• <b>destination (string)</b>: The destination. The supported values are: the employeeId of an agent, the name of an AgentGroup, the name of a Skill, the name of an InteractionQueue, the name of a RoutingPoint, and a phone number for CustomContact.</td></tr>
<tr><td>userData</td><td>object</td><td>The attached user data key/value object. Set an undefined or empty JSON object if you don't want to set any user data.</td></tr>
<tr><td>extensions</td><td>object</td><td>The extensions key/value object. Set an undefined or empty JSON</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>singleStepConference(<i>interactionId, targetQuery, userData, extensions, succeeded, failed</i>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>object if you don't want to set any extensions. This is not applicable for the chat media.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

singleStepTransfer

<table>
<tr><th>Signature</th><th>singleStepTransfer(<i>interactionId, targetQuery, userData, extensions, succeeded, failed</i>)</th></tr>
<tr><th>Description</th><td>Make a single step transfer.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr>
<tr><td>targetQuery</td><td>object or string</td><td>The destination target object, or a character string (for example, phone number).<ul><li>If targetQuery is a character string, the Service Client API creates the</li></ul></td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>singleStepTransfer(<i>interactionId, targetQuery, userData, extensions, succeeded, failed</i>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>operation that uses a target of type <b>CustomContact</b> with a destination set to this value.
• If targetQuery is a JSON object, specify the following sub parameters:
• <b>type (string)</b>: The target type. The possible values are: "AGENT", "AGENT_GROUP", "SKILL", "INTERACTION_QUEUE", "ROUTING_POINT", and "CUSTOM_CONTACT".
• <b>destination (string)</b>: The destination. The supported values are: the employeeId of an Agent, the name of an</td></tr>
</table>
</td></tr>
</table>

Signature	singleStepTransfer(*interactionId, targetQuery, userData, extensions, succeeded, failed*)
	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>AgentGroup, the name of a Skill, the name of an InteractionQueue, the name of a RoutingPoint, and a phone number for CustomContact.</td></tr><tr><td>userData</td><td>object</td><td>The attached user data key/value object. Set an undefined or empty JSON object if you don't want to set any user data.</td></tr><tr><td>extensions</td><td>object</td><td>The extensions key/value object. Set an undefined or empty JSON object if you don't want to set any extensions.</td></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table>

unblockMarkdone

Signature	unblockMarkdone(*interactionId*, succeeded, failed)
Description	Unblock the mark done operation on the selected interaction that was previously blocked.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>interactionId</td><td>string</td><td>The unique interaction identifier of the interaction to prevent the mark done operation.</td></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table>

accept

Signature	accept(*interactionId*, *succeeded*, *failed*)
Description	Accept an interaction when it is ringing in Agent Workspace.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>interactionId</td><td>string</td><td>The unique interaction identifier of the interaction to be accepted.</td></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table>

reject

<table>
<tr><th>Signature</th><td>reject(<i>interactionId</i>, <i>succeeded</i>, <i>failed</i>)</td></tr>
<tr><th>Description</th><td>Reject an interaction when it is ringing in Agent Workspace.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique interaction identifier of the interaction to be rejected.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

Type definitions

The Interaction namespace includes the following object types:

- Interaction
- Party
- Contact

Interaction

<table>
<tr><th>Description</th><td>Represents the JSON structure of an interaction. Attributes specific to voice interactions are: <code>callUuid</code>, <code>direction</code>, <code>callType</code>, <code>ani</code>, <code>dnis</code> and <code>recordingState</code>.</td></tr>
<tr><th>Type</th><td>Object</td></tr>
<tr><th>Properties</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction. <b>Note:</b> This is a client-side ID that is lost</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Description</th><th>Represents the JSON structure of an interaction. Attributes specific to voice interactions are: <code>callUuid</code>, <code>direction</code>, <code>callType</code>, <code>ani</code>, <code>dnis</code> and <code>recordingState</code>.</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>on the next session or refresh.</td></tr>
<tr><td>parentInteractionId</td><td>string</td><td>The unique identifier for the parent interaction. <b>Note:</b> This is a client-side ID that is lost on the next session or refresh.</td></tr>
<tr><td>caseId</td><td>string</td><td>This identifier targets the case that this interaction is part of.</td></tr>
<tr><td>userData</td><td>object</td><td>The attached user data key/value object that is updated with each interaction event.</td></tr>
<tr><td>state</td><td>string</td><td>The current state of the interaction. Possible values are:
• UNKNOWN — An unknown state.
• IDLE — Specifies a non-active interaction which could be closed.
• RINGING — The inbound</td></tr>
</table>
</td></tr>
</table>

Description	Represents the JSON structure of an interaction. Attributes specific to voice interactions are: `callUuid`, `direction`, `callType`, `ani`, `dnis` and `recordingState`.
	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>call is ringing.<ul><li>DIALING — The outbound call is ringing.</li><li>TALKING — The call is established.</li><li>HELD — The call is on hold.</li><li>PREVIEW — The interaction is a call preview.</li><li>INVITED — The open media interaction is inviting.</li><li>ACCEPTED — The open media interaction is accepted.</li><li>CREATED — The open media interaction has been created.</li><li>PULLED — The open media interaction has been pulled</li></ul></td></tr></table>

<table>
<tr><th>Description</th><td>Represents the JSON structure of an interaction. Attributes specific to voice interactions are: `callUuid`, `direction`, `callType`, `ani`, `dnis` and `recordingState`.</td></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>from a workbin.
• REVOKED — The open media interaction has been revoked.
• COMPLETED — The open media interaction has been completed (Mark as done).
• ERROR — The open media interaction has an error.
• SAVED — The open media interaction has been saved.
• TRANSFERRING — The open media interaction is being transferred.
• TRANSFER_COMPLETED — The open media interaction has been transferred</td></tr>
</table>
</td></tr>
</table>

Description	Represents the JSON structure of an interaction. Attributes specific to voice interactions are: `callUuid`, `direction`, `callType`, `ani`, `dnis` and `recordingState`.
	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>and the transfer has been completed. • INVITED_CONFERENCE — The open media interaction receives a conference invitation. • LEFT_CONFERENCE — The open media interaction has left the conference. • USER_DATA_ATTACHED — Data has been attached to the interaction. • USER_DATA_UPDATED — The attached data has changed in the interaction. • JOIN_PENDING — Trying to join the chat session. • JOIN_FAILED — The connection with the chat server</td></tr></table>

Description	Represents the JSON structure of an interaction. Attributes specific to voice interactions are: `callUuid`, `direction`, `callType`, `ani`, `dnis` and `recordingState`.

Name	Type	Description
		failed. • HISTORY_IN_PROGRESS — Loading the content of the chat interaction. • HISTORY_DONE — The content of the chat interaction has been loaded. • CANCELLED — The outbound email is cancelled. • SENT — The outbound email is sent. • READY — The call preview is ready. • CANCELED — The call preview is cancelled. • REJECTED — The call preview is rejected.
previousState	string	The previous state of the interaction.
parties	Array.	A collection of all the parties involved in

<table>
<tr><th>Description</th><th>Represents the JSON structure of an interaction. Attributes specific to voice interactions are: <code>callUuid</code>, <code>direction</code>, <code>callType</code>, <code>ani</code>, <code>dnis</code> and <code>recordingState</code>.</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>the interaction.</td></tr>
<tr><td>isConsultation</td><td>boolean</td><td>This property is true if the interaction is a consultation; otherwise, it's false.</td></tr>
<tr><td>isMainCaseInteraction</td><td>boolean</td><td>This property is true if the interaction is the main interaction in the customer case; otherwise, it's false. In Workspace Web Edition, the main interaction is related to Case Information, Disposition, Note, Contact Profile, and so on.</td></tr>
<tr><td>callUuid</td><td>string</td><td>The UUID of the call. This attribute is only on voice interactions.</td></tr>
<tr><td>direction</td><td>string</td><td>The call direction. Possible values are: IN, OUT or UNKNOWN. This attribute is only on voice interactions.</td></tr>
<tr><td>callType</td><td>string</td><td>The call type. Possible values are:</td></tr>
</table>
</td></tr>
</table>

Description	Represents the JSON structure of an interaction. Attributes specific to voice interactions are: `callUuid`, `direction`, `callType`, `ani`, `dnis` and `recordingState`.

Name	Type	Description
		INTERNAL, INBOUND, OUTBOUND, CONSULT or UNKNOWN. This attribute is only on voice interactions.
ani	string	The Automatic Number Identification service. This attribute is only on voice interactions.
dnis	string	The Dialed Number Identification Service. This attribute is only on voice interactions.
recordingState	string	The call recording state. Possible values are: STOPPED, RECORDING or PAUSED. This attribute is only on voice interactions.
isCaseSelected	boolean	Is true if the case containing this interaction is selected, otherwise is false.
ronaCallState	string	This value is populated on event RELEASED when an agent receives

Description	Represents the JSON structure of an interaction. Attributes specific to voice interactions are: `callUuid`, `direction`, `callType`, `ani`, `dnis` and `recordingState`.

Name	Type	Description
		an inbound call and does not answer. Possible values are: REDIRECTED or NO_ANSWER.
isCaseExpanded	boolean	Is true if the case containing this interaction is expanded, otherwise is false.
interactionUUID	string	The `attr_itx_id` for a multimedia interaction or the `callUuid` for a voice interaction.
connId	string	The unique connection ID from the T-Server.
contact	interaction.Contact	An object representing the contact's information.

Party

Description	Represents the JSON structure of a party.
Type	Object
Properties	

Name	Type	Description
name	string	The name of the party.

Contact

<table>
<tr><th>Description</th><td>Represents the JSON structure of a contact.</td></tr>
<tr><th>Type</th><td>Object</td></tr>
<tr><th>Properties</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>displayName</td><td>string</td><td>The contact's display name.</td></tr>
<tr><td>firstNname</td><td>string</td><td>The contact's first name.</td></tr>
<tr><td>lastName</td><td>string</td><td>The contact's last name.</td></tr>
</table>
</td></tr>
</table>

Media namespace

Contents

Learn about the Media namespace methods and type definitions in the Service Client API.

Methods

The Media namespace includes the following methods:

- getMediaList
- getMediaByName
- setState

getMediaList

<table>
<tr><th>Signature</th><td>getMediaList(succeeded, failed) → {Array.}</td></tr>
<tr><th>Description</th><td>Get the list of media with attributes.</td></tr>
<tr><th>Parameters</th><td><table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table></td></tr>
<tr><th>Returns</th><td>Array.</td></tr>
</table>

getMediaByName

<table>
<tr><th>Signature</th><td>getMediaByName(<i>name</i>, succeeded, failed)</td></tr>
<tr><th>Description</th><td>Get the media attributes.</td></tr>
<tr><th>Parameters</th><td><table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>name</td><td>string</td><td>The media name.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Signature</th><th>getMediaByName(<em>name</em>, succeeded, failed)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

setState

<table>
<tr><th>Signature</th><th>setState(<em>name</em>, <em>stateOperationName</em>, succeeded, failed)</th></tr>
<tr><th>Description</th><td>Sets the media state.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>name</td><td>string</td><td>The media name.</td></tr>
<tr><td>stateOperationName</td><td>string</td><td>An <code>operationName</code> from the agent states list. See State.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

Type definitions

The Media namespace includes the following object types:

- Media
- State
- Device

Media

Description	Represents the JSON structure of a media.
Type	Object

<table>
<tr><th>Description</th><td>Represents the JSON structure of a media.</td></tr>
<tr><th>Properties</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>name</td><td>string</td><td>The media name.</td></tr>
<tr><td>state</td><td>media.State</td><td>The media state object.</td></tr>
</table>
</td></tr>
</table>

State

<table>
<tr><th>Description</th><td>Represents the JSON structure of a media state.</td></tr>
<tr><th>Type</th><td>Object</td></tr>
<tr><th>Properties</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>type</td><td>string</td><td>The type of operation. Possible values are:
- LOGOUT
- READY
- PARTIAL_READY *
- NOT_READY
- NOT_READY_ACTION_CODE
- NOT_READY_AFTER_CALLWO
- NOT_READY_AFTER_CALLWO
- DND_ON
- OUT_OF_SERVICE *
- LOGOUT_DND_ON *
- UNKNOWN *</td></tr>
<tr><td>displayName</td><td>string</td><td>The display name of the state.</td></tr>
<tr><td>operationName</td><td>string</td><td>The operation name to use with agent.setState and media.setState.</td></tr>
</table>
</td></tr>
</table>

** States that are limited to an event and can't be applied by code*

Device

<table>
<tr><th>Description</th><td>Represents the JSON structure of a media.</td></tr>
<tr><th>Type</th><td>Object</td></tr>
<tr><th>Properties</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>number</td><td>string</td><td>The phone number configured for an agent – the physical DN.
<b>Note</b>: This property is applicable only for voice data.</td></tr>
<tr><td>dynamicPhoneNumber</td><td>string</td><td>The dynamic phone number configured for the agent for the session.
<b>Note</b>: This property is applicable only for voice data. This property is applicable only when there is an alternate phone number and applicable for the current session only.</td></tr>
</table>
</td></tr>
</table>

System namespace

Contents

Learn about the System namespace methods in the Service Client API.

Important

Depending on your environment, you might need to contact your Genesys representative to complete the configuration described on this page.

Methods

The System namespace includes the following methods:

- amIVisible
- closeDialog
- closeToast
- closeViewInApplicationMenuBar
- getAllowedServices
- isFrameLeading
- isFrameFollowing
- isFrameNegotiating
- isFrameLeadingOrNegotiating
- isLastActiveFrame
- popupToast
- openDialog
- triggerActivity
- updateViewInApplicationMenuBar
- updateToast

amIVisible

Signature	amIVisible(succeeded, failed) → {boolean}
Description	Get the current visibility state of the frame.

<table>
<tr><th>Signature</th><th>amIVisible(succeeded, failed) → {boolean}</th></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
<tr><td><b>Returns</b></td><td><code>true</code> if the frame is visible.</td></tr>
</table>

Sample request

```
setTimeout(function() {
    genesys.wwe.service.system.amIVisible(succeeded, failed);
}, 3000); // This gives 3 seconds to switch the panel to test.
```

Sample response

The asynchronous answer is included in the `data` attribute:

```
{
  "request": "system.amIVisible",
  "data": true,
  "userAgent": "WWE Server",
  "protocolVersion": 2
}
```

closeDialog

<table>
<tr><th>Signature</th><th>closeDialog(<i>dialogId</i>, <i>succeeded</i>, <i>failed</i>) → {boolean}</th></tr>
<tr><td><b>Description</b></td><td>Close a previously opened dialog.</td></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>dialogId</td><td>string</td><td>The dialog identifier (returned in the response of openDialog).</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

Signature	closeDialog(*dialogId*, *succeeded*, *failed*) → {boolean}
Returns	`true` if the dialog is closed; `false` if the dialog is not found.

closeToast

Signature	closeToast(*id*, succeeded, failed) → {boolean}
Description	Closes the specified toast.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>id</td><td>string</td><td>The identifier of the toast to close. The identifier is returned by the popupToast method.</td></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table>
Returns	`true` if the toast has been updated; `false` if the toast identifier has not been found.

closeViewInApplicationMenuBar

Signature	closeViewInApplicationMenuBar(parameters, succeeded, failed) → {boolean}
Description	Removes the given view from the **Application Menu** bar region.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>name</td><td>string</td><td>The name of the custom view to be removed.</td></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function</td></tr></table>

Signature	closeViewInApplicationMenuBar(parameters, succeeded, failed) → {boolean}
	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>called when the operation fails.</td></tr></table>
Returns	`true` if the view is removed; `false` if the view name is not found.

Sample request

```
genesys.wwe.service.system.closeViewInApplicationMenuBar("view1", succeeded, failed)
```

Sample response

```
{
   "request": "system.closeViewInApplicationMenuBar",
    "data": true,
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

Sample request

```
genesys.wwe.service.system.closeDialog("wweCustomDialog1", succeeded, failed)
```

Sample response

The asynchronous answer is included in the `data` attribute:

```
{
    "request": "system.closeDialog",
    "data": true,
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

getAllowedServices

Signature	getAllowedServices(succeeded, failed) → {Array.}
Description	Gets the list of allowed services, as determined by the Security configuration. If the domain of the web application that calls this method isn't listed in the service-client-api.accepted-web-content-origins option, then this method fails.
Parameters	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr></table>

Signature	getAllowedServices(succeeded, failed) → {Array.}		
	Name	**Type**	**Description**
	failed	function	A function called when the operation fails.
Returns	Array.		

isFrameLeading

Signature	isFrameLeading(*succeeded*, *failed*) → {boolean}		
Description	Find out if the browser tab is leading.		
Parameters	**Name**	**Type**	**Description**
	succeeded	function	A function called when the operation succeeds.
	failed	function	A function called when the operation fails.
Returns	`true` if the browser tab is the leader.		

isFrameFollowing

Signature	isFrameFollowing(*succeeded*, *failed*) → {boolean}		
Description	Find out if the browser tab is following.		
Parameters	**Name**	**Type**	**Description**
	succeeded	function	A function called when the operation succeeds.
	failed	function	A function called when the operation fails.
Returns	`true` if this browser tab is following.		

isFrameNegotiating

<table>
<tr><th>Signature</th><td>isFrameNegotiating(<i>succeeded</i>, <i>failed</i>) → {boolean}</td></tr>
<tr><th>Description</th><td>Find out if there is an election in progress and the browser tab state is not yet set to leading or following (the tab is "negotiating.")</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
<tr><th>Returns</th><td><code>true</code> if the tab is negotiating.</td></tr>
</table>

isFrameLeadingOrNegotiating

<table>
<tr><th>Signature</th><td>isFrameLeadingOrNegotiating(<i>succeeded</i>, <i>failed</i>) → {boolean}</td></tr>
<tr><th>Description</th><td>Find out if the browser tab is leading or there is an election in progress and the tab state is not yet set to leading or following (the tab is "negotiating.").</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
<tr><th>Returns</th><td><code>true</code> if the browser tab is leading or negotiating.</td></tr>
</table>

isLastActiveFrame

<table>
<tr><th>Signature</th><td>isLastActiveFrame(<i>succeeded</i>, <i>failed</i>) → {boolean}</td></tr>
<tr><th>Description</th><td>Find out if this is the last active browser tab.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>isLastActiveFrame(<em>succeeded</em>, <em>failed</em>) → {boolean}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
<tr><td><strong>Returns</strong></td><td>true if this is the last active browser tab.</td></tr>
</table>

openDialog

<table>
<tr><th>Signature</th><th>openDialog(<em>url</em>, <em>options</em>, <em>succeeded</em>, <em>failed</em>) → {string}</th></tr>
<tr><td><strong>Description</strong></td><td>Open an iframe in a dialog, based on the configured parameters.</td></tr>
<tr><td><strong>Parameters</strong></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>url</td><td>string</td><td>The URL of the iframe to load in the dialog.</td></tr>
<tr><td>options</td><td>object</td><td>Optional parameters to configure the dialog. This value can't be null, so you must pass {} if there are no specific options. You can include any of the following options:
• label - Set a custom value for the aria-label attribute on the dialog. When the dialog pops up, this value identifies it</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>openDialog(<i>url</i>, <i>options</i>, <i>succeeded</i>, <i>failed</i>) → {string}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>to accessibility tools like screen readers.
• width - The initial width of the dialog. Valid formats are px or %.
• height - The initial height of the dialog. Valid formats are px or %.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
<tr><td><b>Returns</b></td><td>The dialog identifier or null if the <code>url</code> parameter is not defined.</td></tr>
</table>

Sample request

```
genesys.wwe.service.system.openDialog("", {
  label: "Dialog $Agent.FullName$",
  width: "430px",
  height: "325px"
}, succeeded, failed)
```

Sample response

The asynchronous answer is included in the `data` attribute:

```
{
    "request": "system.openDialog",
    "data": "wweCustomDialog1",
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

popupToast

<table>
<tr><th>Signature</th><th>popupToast(<i>parameters</i>, succeeded, failed) → {string}</th></tr>
<tr><th>Description</th><td>Pops up a new custom toast.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>parameters</td><td>object</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>title</td><td>string</td><td>The title</td></tr>
<tr><td>iconUrl</td><td>string</td><td>The URL of the icon you want to display in the title bar of the custom toast popup.</td></tr>
<tr><td>subject</td><td>string</td><td>Optional. The subject</td></tr>
<tr><td>message</td><td>string</td><td>Optional. The message</td></tr>
<tr><td>keyValues</td><td>string</td><td>Optional. JSON object used to fill the key value pair list. For example:</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>popupToast(<em>parameters</em>, succeeded, failed) → {string}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td><code>{"key1" ; "value one","key2" ; "value two","key3" ; "value three"}</code>.</td></tr>
<tr><td>buttons</td><td>Array</td><td>Optional. Each character string in this array becomes a button. All buttons are displayed as buttons, not hyperlinks, in the following order: [Button 2] [Button 3] ... [Button N] [Button 1].</td></tr>
<tr><td>buttonShowDismiss</td><td>boolean</td><td>Optional. If set to <code>true</code>, displays the <strong>Show</strong> and <strong>Dismiss</strong></td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>popupToast(<em>parameters</em>, succeeded, failed) → {string}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>buttons and pops up the current iframe if the <strong>Show</strong> button is pushed. If set to <code>false</code>, displays '"OK"' or custom buttons based on the parameter's buttons.</td></tr>
<tr><td>autoCloseTimeout</td><td>Object</td><td>Optional. If set to greater than <code>0</code>, the popup is automatically closed after the specified milliseconds.</td></tr>
<tr><td>sendToMyMessage</td><td>Object</td><td>Optional. If set to</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>popupToast(<i>parameters</i>, succeeded, failed) → {string}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td><code>true</code>, sends the <b>subject</b>, <b>iconUrl</b>, <b>title</b>, <b>keyValues</b>, and <b>message</b> parameters to the <b>MyMessage</b> panel.</td></tr>
<tr><td>width</td><td>number</td><td>Optional. The width of the custom toast popup, in pixels. This values takes precedence over the service-client-api.toast.width configuration option.</td></tr>
</table>
</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
<tr><td><b>Returns</b></td><td>A unique identifier</td></tr>
</table>

triggerActivity

<table>
<tr><th>Signature</th><td>triggerActivity(succeeded, failed)</td></tr>
<tr><th>Description</th><td>Triggers a fake activity to prevent the inactivity timer from closing the agent session.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

updateViewInApplicationMenuBar

<table>
<tr><th>Signature</th><td>updateViewInApplicationMenuBar(parameters, succeeded, failed) → {string}</td></tr>
<tr><th>Description</th><td>Creates a custom view in the <b>Application Menu bar</b> region.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>name</td><td>string</td><td>A unique name for the custom view of the <b>Application Menu bar</b> that is to be created or updated. If a view with the given name already exists, it will be updated, otherwise, a new view will be created.</td></tr>
<tr><td>iconUrl</td><td>string</td><td>The URL of the icon you want to display in the custom view of the <b>Application Menu bar</b> region. This parameter is</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>updateViewInApplicationMenuBar(parameters, succeeded, failed) → {string}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>mandatory if <b>label</b> is not provided.</td></tr>
<tr><td>label</td><td>string</td><td>The main textual content to be displayed for the custom view. This parameter is mandatory if <b>iconUrl</b> is not provided.</td></tr>
<tr><td>shortLabel</td><td>string</td><td>Optional. A shorter version of the <b>label</b> that will be used in the shortened mode if <b>iconUrl</b> is not available.</td></tr>
<tr><td>tooltip</td><td>string</td><td>Optional. The tooltip content to be shown when the mouse is hovered on the custom view.</td></tr>
<tr><td>labelColor</td><td>string</td><td>Optional. The color of the label text in case-insensitive hex color code format, for example, #FFFFF.</td></tr>
<tr><td>backgroundColor</td><td>string</td><td>Optional. The background color of the region where icon and title are displayed. The format of the background color is case-</td></tr>
</table>
</td></tr>
</table>

Signature	updateViewInApplicationMenuBar(parameters, succeeded, failed) → {string}
	<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>insensitive hex color code, for example, #FFFFF. By default, it is usually the same color as the navigation bar.</td></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table>
Returns	View name if successful.

Sample request

```
genesys.wwe.service.system.updateViewInApplicationMenuBar({
    name: "view1",
    iconUrl: "https://cdn1.iconfinder.com/data/icons/free-social-media-12/32/
RSS_social_media-128.png",
    label: "Main content text",
    shortLabel: "Short text"
    tooltip: "Tooltip text",
    labelColor: "#FFFFF",
    backgroundColor: "#00000"
}, succeeded, failed)'
```

Sample response

```
{
    "request": "system.updateViewInApplicationMenuBar",
    "data": "view1",
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

updateToast

Signature	updateToast(*id*, *parameters*, succeeded, failed) → {boolean}
Description	Updates the specified toast.

<table>
<tr><th>Signature</th><th>updateToast(<i>id</i>, <i>parameters</i>, succeeded, failed) → {boolean}</th></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>id</td><td>string</td><td>The identifier of the toast to update. The identifier is returned by the popupToast method.</td></tr>
<tr><td>parameters</td><td>object</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>title</td><td>string</td><td>The title</td></tr>
<tr><td>iconUrl</td><td>string</td><td>The URL of the icon you want to display in the title bar of the custom toast popup.</td></tr>
<tr><td>subject</td><td>string</td><td>Optional. The subject.</td></tr>
<tr><td>message</td><td>string</td><td>Optional. The message.</td></tr>
<tr><td>keyValues</td><td>object</td><td>Optional. JSON object used to fill the key value pair list. For</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>updateToast(<em>id</em>, <em>parameters</em>, succeeded, failed) → {boolean}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>example: <code>{"key1" : "value one","key2" : "value two","key3" : "value three"}</code>.</td></tr>
<tr><td>buttons</td><td>Array</td><td>Each character string in this array becomes a button. All buttons are displayed as buttons, not hyperlinks, in the following order: [Button 2] [Button 3] ... [Button N] [Button 1].</td></tr>
<tr><td>buttonShowDismiss</td><td>boolean</td><td>If set to <code>true</code>, displays <strong>Show</strong> and <strong>Dismiss</strong> buttons and</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>updateToast(<em>id</em>, <em>parameters</em>, succeeded, failed) → {boolean}</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>pops up the current iframe if the <strong>Show</strong> button is pushed. If set to <code>false</code>, displays '"OK"' or custom buttons based on the parameter's buttons.</td></tr>
</table>
</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
<tr><th>Returns</th><td><code>true</code> if the toast has been updated; <code>false</code> if the toast identifier has not been found.</td></tr>
</table>

Voice namespace

Contents

Learn about the Voice namespace methods in the Service Client API.

Methods

The Voice namespace includes the following methods:

- answer
- dial
- dialEx
- hangUp
- hold
- resume
- pauseCallRecording
- resumeCallRecording
- startCallRecording
- stopCallRecording
- isMicrophoneMute
- muteMicrophone
- unmuteMicrophone
- isSpeakerMute
- muteSpeaker
- unmuteSpeaker

answer

<table>
<tr><th>Signature</th><td>answer('interactionId', succeeded, failed)</td></tr>
<tr><th>Description</th><td>Answers the incoming call.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td></td><td>The interaction identifier</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>answer('interactionId', succeeded, failed)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td>called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

dial

<table>
<tr><th>Signature</th><th>dial(destination, userData, succeeded, failed)</th></tr>
<tr><td><b>Description</b></td><td>Calls the destination in the same way Workspace Web Edition calls the destination from Team Communicator.</td></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>destination</td><td>string</td><td></td><td>The call destination number.</td></tr>
<tr><td>userData</td><td>object</td><td></td><td>The attached user data key/value object that is updated with each interaction event.</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

dialEx

<table>
<tr><th>Signature</th><td>dialEx(<i>destination</i>, <i>userData</i>, <i>extensions</i>, <i>succeeded</i>, <i>failed</i>)</td></tr>
<tr><th>Description</th><td>Calls the destination with the attached data and extensions.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>destination</td><td>string</td><td></td><td>The call destination number.</td></tr>
<tr><td>userData</td><td>object</td><td></td><td>The attached user data key/value object. Set an undefined or empty JSON object if you don't want to set any user data.</td></tr>
<tr><td>extensions</td><td>object</td><td></td><td>The extensions key/value object. Set an undefined or empty JSON object if you don't want to set any extensions.</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

hangUp

<table>
<tr><th>Signature</th><td>hangUp('interactionId', succeeded, failed)</td></tr>
<tr><th>Description</th><td>Releases the incoming call.</td></tr>
<tr><th>Parameters</th><td><table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td></td><td>The interaction identifier</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table></td></tr>
</table>

hold

<table>
<tr><th>Signature</th><td>hold('interactionId', succeeded, failed)</td></tr>
<tr><th>Description</th><td>Holds the incoming call.</td></tr>
<tr><th>Parameters</th><td><table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td></td><td>The interaction identifier</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table></td></tr>
</table>

resume

<table>
<tr><th>Signature</th><td>resume('interactionId', succeeded, failed)</td></tr>
<tr><th>Description</th><td>Resumes the held call.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td></td><td>The interaction identifier</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

pauseCallRecording

<table>
<tr><th>Signature</th><td>pauseCallRecording('interactionId', succeeded, failed)</td></tr>
<tr><th>Description</th><td>Pause the call recording.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td></td><td>The interaction identifier</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

resumeCallRecording

<table>
<tr><th>Signature</th><td>resumeCallRecording('interactionId', succeeded, failed)</td></tr>
<tr><th>Description</th><td>Resumes the call recording.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td></td><td>The interaction identifier</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

startCallRecording

<table>
<tr><th>Signature</th><td>startCallRecording('interactionId', succeeded, failed)</td></tr>
<tr><th>Description</th><td>Starts the call recording.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td></td><td>The interaction identifier</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

stopCallRecording

<table>
<tr><th>Signature</th><td>stopCallRecording('interactionId', succeeded, failed)</td></tr>
<tr><th>Description</th><td>Stops the call recording.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td></td><td>The interaction identifier</td></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

isMicrophoneMute

<table>
<tr><th>Signature</th><td>isMicrophoneMute(succeeded, failed)</td></tr>
<tr><th>Description</th><td>Get the mute state of the microphone of the SIP Endpoint.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

muteMicrophone

<table>
<tr><th>Signature</th><td>muteMicrophone(succeeded, failed)</td></tr>
<tr><th>Description</th><td>Mute the microphone of the SIP Endpoint.</td></tr>
</table>

<table>
<tr><th>Signature</th><td>muteMicrophone(succeeded, failed)</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

unmuteMicrophone

<table>
<tr><th>Signature</th><td>unmuteMicrophone(succeeded, failed)</td></tr>
<tr><th>Description</th><td>Unmute the microphone of the SIP Endpoint.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

isSpeakerMute

<table>
<tr><th>Signature</th><td>isSpeakerMute(succeeded, failed)</td></tr>
<tr><th>Description</th><td>Get the mute state of the speaker of the SIP Endpoint.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>isSpeakerMute(succeeded, failed)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

muteSpeaker

<table>
<tr><th>Signature</th><th>muteSpeaker(succeeded, failed)</th></tr>
<tr><th>Description</th><td>Mute the speaker of the SIP Endpoint.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

unmuteSpeaker

<table>
<tr><th>Signature</th><th>unmuteSpeaker(succeeded, failed)</th></tr>
<tr><th>Description</th><td>Unmute the speaker of the SIP Endpoint.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td></td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td></td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

Outbound namespace

Contents

- Developer

Learn about the Outbound namespace methods in the Service Client API.

Important

Depending on your environment, you might need to contact your Genesys representative to complete the configuration described on this page.

Methods

The Outbound namespace includes the following methods:

- getCampaigns
- getPreviewRecord
- callPreviewRecord
- rejectPreviewRecord
- cancelPreviewRecord
- startDirectPushPreview
- stopDirectPushPreview
- getListOfCallResults
- setCallResult
- getCallResult
- setDoNotCall
- removeDoNotCall
- rescheduleRecord
- cancelReschedule
- getChainedRecords
- getRecordFields
- updateRecordFields

getCampaigns

<table>
<tr><th>Signature</th><td>getCampaigns(<i>succeeded</i>, <i>failed</i>)</td></tr>
<tr><th>Description</th><td>Get the details of all outbound campaigns (loaded or active) for the current agent.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

getPreviewRecord

<table>
<tr><th>Signature</th><td>getPreviewRecord(<i>campaignName</i>, <i>succeeded</i>, <i>failed</i>)</td></tr>
<tr><th>Description</th><td>Get a preview record from Outbound Contact Server.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>campaignName</td><td>string</td><td>The name of the outbound campaign.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

callPreviewRecord

<table>
<tr><th>Signature</th><td>callPreviewRecord(<i>interactionId</i>, <i>recordHandle</i>, <i>succeeded</i>, <i>failed</i>)</td></tr>
<tr><th>Description</th><td>Make a call using the preview record.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>callPreviewRecord(<i>interactionId</i>, <i>recordHandle</i>, <i>succeeded</i>, <i>failed</i>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>interaction.</td></tr>
<tr><td>recordHandle</td><td>number</td><td>The record number in the chain to be dialed.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

rejectPreviewRecord

<table>
<tr><th>Signature</th><th>rejectPreviewRecord(<i>succeeded</i>, <i>failed</i>)</th></tr>
<tr><td><b>Description</b></td><td>Reject a pull preview, push preview, or direct push preview record.</td></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

cancelPreviewRecord

<table>
<tr><th>Signature</th><th>cancelPreviewRecord(<i>succeeded</i>, <i>failed</i>)</th></tr>
<tr><td><b>Description</b></td><td>Cancel a pull preview, push preview, or direct push preview record.</td></tr>
<tr><td><b>Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>cancelPreviewRecord(<em>succeeded</em>, <em>failed</em>)</th></tr>
<tr><td></td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>called when the operation fails.</td></tr></table></td></tr>
</table>

startDirectPushPreview

<table>
<tr><th>Signature</th><th>startDirectPushPreview(<em>succeeded</em>, <em>failed</em>)</th></tr>
<tr><td><strong>Description</strong></td><td>Send a Dialing Mode Start request to Outbound Contact Server to start sending direct push preview records to the agent.</td></tr>
<tr><td><strong>Parameters</strong></td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table></td></tr>
</table>

stopDirectPushPreview

<table>
<tr><th>Signature</th><th>stopDirectPushPreview(<em>succeeded</em>, <em>failed</em>)</th></tr>
<tr><td><strong>Description</strong></td><td>Send a Dialing Mode Stop request to Outbound Contact Server to stop sending direct push preview records to the agent.</td></tr>
<tr><td><strong>Parameters</strong></td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr><tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr></table></td></tr>
</table>

getListOfCallResults

Signature	getListOfCallResults(*succeeded*, *failed*)
Description	Get the list of call results currently available in Workspace Web Edition.

<table>
<tr><th>Signature</th><th>getListOfCallResults(<em>succeeded</em>, <em>failed</em>)</th></tr>
<tr><td><strong>Parameters</strong></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

Sample request

```
genesys.wwe.service.outbound.getListOfCallResults(succeeded, failed)
```

Sample response

```
{
    "request": "outbound.getListOfCallResults",
    "data": {
        "OK": 0,
        "GENERAL_ERROR": 3,
        "SYSTEM_ERROR": 4,
        "BUSY": 6,
        "NO_ANSWER": 7,
        "SIT_DETECTED": 8,
        "ANSWERING_MACHINE": 9,
        "ALL_TRUNKS_BUSY": 10,
        "SIT_INVALID_NUM": 11,
        "SIT_VACANT": 12,
        "SIT_OPERINTERCEPT": 13,
        "SIT_UNKNOWN": 14,
        "SIT_NO_CIRCUIT": 15,
        "SIT_REORDER": 16,
        "FAXDETECTED": 17,
        "ABANDONED": 21,
        "DROPPED": 26,
        "DROPPED_NO_ANSWER": 27,
        "UNKNOWN": 28,
        "SILENCE": 32,
        "ANSWER": 33,
        "NUTONE": 34,
        "NO_DIAL_TONE": 35,
        "NO_PROGRESS": 36,
        "NO_RINGBACK": 37,
        "NO_ESTABLISHED": 38,
        "PAGER_DETECTED": 39,
        "WRONG_PARTY": 40,
        "DIAL_ERROR": 41,
        "CALL_DROP_ERROR": 42,
        "SWITCH_ERROR": 43,
        "NO_FREE_PORT_ERROR": 44,
        "TRANSFER_ERROR": 45,
        "STALE": 46,
        "AGENT_CALLBACK_ERROR": 47,
        "GROUP_CALLBACK_ERROR": 48,
        "DO_NOT_CALL": 51,
        "CANCEL_RECORD": 52,
        "WRONG_NUMBER": 53
    },
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

setCallResult

<table>
<tr><th>Signature</th><td>setCallResult(<i>interactionId</i>, <i>callResult</i>, <i>succeeded</i>, <i>failed</i>)</td></tr>
<tr><th>Description</th><td>Set the call result for this interaction.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction. The interaction should have an active or completed call. "Do Not Call" must not be set for the interaction.</td></tr>
<tr><td>callResult</td><td>string</td><td>The call result value, which must be a number.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

getCallResult

<table>
<tr><th>Signature</th><td>getCallResult(<i>interactionId</i>, <i>succeeded</i>, <i>failed</i>)</td></tr>
<tr><th>Description</th><td>Get the call result already set in an outbound record, if any.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>getCallResult(<i>interactionId</i>, <i>succeeded</i>, <i>failed</i>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>interaction.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

Sample request

```
genesys.wwe.service.outbound.getCallResult(interactionId, succeeded, failed)
```

Sample response

```
{
    "request": "outbound.getCallResult",
    "data": 6,
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

setDoNotCall

<table>
<tr><th>Signature</th><th>setDoNotCall(<i>interactionId</i>, <i>succeeded</i>, <i>failed</i>)</th></tr>
<tr><th>Description</th><td>Set the interaction to "Do Not Call".</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction. The interaction should have an active or completed call.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

removeDoNotCall

Signature	removeDoNotCall(*interactionId*, *succeeded*, *failed*)
Description	Remove "Do Not Call" from the interaction.
Parameters	

Name	Type	Description
interactionId	string	The unique identifier for the interaction. The interaction should have an active or completed call.
succeeded	function	A function called when the operation succeeds.
failed	function	A function called when the operation fails.

rescheduleRecord

Signature	rescheduleRecord(*interactionId*, *recordHandle*, *rescheduleDate*, *callbackType*, *succeeded*, *failed*)
Description	Set the schedule information on the record based on its time zone. You can perform this operation regardless of how the Workspace Web Edition options privilege.outbound.can-reschedule and privilege.outbound.can-reschedule-before-call are configured.
Parameters	

Name	Type	Description
interactionId	string	The unique identifier for the interaction. Note: For Preview and Push Preview modes, once the call is made the ID provided becomes the new

<table>
<tr><th>Signature</th><th>rescheduleRecord(<i>interactionId</i>, <i>recordHandle</i>, <i>rescheduleDate</i>, <i>callbackType</i>, <i>succeeded</i>, <i>failed</i>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>interaction ID that corresponds to the call.</td></tr>
<tr><td>recordHandle</td><td>number</td><td>The record number in the chain to be dialed.</td></tr>
<tr><td>rescheduleDate</td><td>string</td><td>The date for which the callback is to be rescheduled, in MM/DD/YYYY HH:MM format. This date should be in the time zone of the record that is being rescheduled. This ensures the date is set correctly in cases where the agent and the customer are in different time zones. To calculate the correct hour and minute values, you can get the outbound record's time zone offset value from any of the interaction's events.
<b>Example</b>
An agent calls a customer and they ask to be called back one hour later. The agent and</td></tr>
</table>
</td></tr>
</table>

Signature	rescheduleRecord(*interactionId*, *recordHandle*, *rescheduleDate*, *callbackType*, *succeeded*, *failed*)

Name	Type	Description
		customer have the following time zone information: • Agent's time zone - BST • Agent's current time - 2:30 PM • Customer's time zone - EDT • Customer's current time - 9:30 AM In this case, you would make the **rescheduleRecord** request with the **rescheduleDate** HH:MM set to a value of 10:30 and not 15:30.
callbackType	string	The type of callback. Valid values are CAMPAIGN or PERSONAL.
succeeded	function	A function called when the operation succeeds.
failed	function	A function called when the operation fails.

Sample request

```
genesys.wwe.service.outbound.rescheduleRecord('1', 257, '05/27/2021 10:55', 'PERSONAL',
succeeded, failed)
```

cancelReschedule

<table>
<tr><th>Signature</th><td>cancelReschedule(*interactionId*, *succeeded*, *failed*)</td></tr>
<tr><th>Description</th><td>Remove the schedule information from the record.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction. Note: For Preview and Push Preview modes, once the call is made the ID provided becomes the new interaction ID that corresponds to the call.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

getChainedRecords

<table>
<tr><th>Signature</th><td>getChainedRecords(*interactionId*, *succeeded*, *failed*)</td></tr>
<tr><th>Description</th><td>Get the list of chained records for the interaction.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>getChainedRecords(<i>interactionId</i>, <i>succeeded</i>, <i>failed</i>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>the operation fails.</td></tr>
</table>
</td></tr>
</table>

Sample request

```
genesys.wwe.service.outbound.getChainedRecords('1', succeeded, failed)
```

Sample response

```
{
    "request": "outbound.getChainedRecords",
    "data": [
        {
            "records": [
                {
                    Custom_Character: "c"
                    Custom_Datetime: "2021-03-17 14:42:39"
                    Custom_Float: "16.64"
                    Custom_Integer: 0
                    Custom_String_with_default: "Hi there!"
                    Custom_VarChar: ""
                    GSW_AGENT_ID: "+33298025000"
                    GSW_APPLICATION_ID: 139
                    GSW_ATTEMPTS: 0
                    GSW_CALLING_LIST: "Calling List Custom"
                    GSW_CALLING_LIST_DBID: 101
                    GSW_CALL_ATTEMPT_GUID: "003DC7H6HG84DBRT1KMIF1TAES000031"
                    GSW_CALL_RESULT: 28
                    GSW_CAMPAIGN_GROUP_DBID: 101
                    GSW_CAMPAIGN_GROUP_DESCRIPTION: ""
                    GSW_CAMPAIGN_GROUP_NAME: "Outbound Campaign Custom@Agent Group Outbound"
                    GSW_CAMPAIGN_NAME: "Outbound Campaign Custom"
                    GSW_CHAIN_ID: 3
                    GSW_CONTACT_MEDIA_TYPE: "voice"
                    GSW_FROM: 0
                    GSW_PHONE: "+33647005"
                    GSW_PHONE_TYPE: 1
                    GSW_RECORD_HANDLE: 283
                    GSW_REFERENCE_ID: 3
                    GSW_SWITCH_DBID: 101
                    GSW_TZ_NAME: "ACT"
                    GSW_TZ_OFFSET: 34200
                    GSW_UNTIL: 86399
                    GSW_USER_EVENT: "PreviewRecord"
                    IW_BundleUid: "27458420-0348-4345-c693-45bd95b5c81f"
                    IW_CaseUid: "a26f59d2-2979-43c5-5c1d-b0757f9ab077"
                    InteractionSubtype: "OutboundNew"
                    InteractionType: "Outbound"
                    WWE_OUTBOUND_CAMP_TYPE: "PreviewRecord"
                },
                {
                    Custom_Character: "c"
                    Custom_Datetime: "2021-03-17 14:42:32"
                    Custom_Float: "51.69"
                    Custom_Integer: 0
                    Custom_String_with_default: "Hello General Kenobi"
                    Custom_VarChar: ""
                    GSW_AGENT_ID: "+33298025000"
                    GSW_APPLICATION_ID: 139
                    GSW_ATTEMPTS: 0
                    GSW_CALLING_LIST: "Calling List Custom"
                    GSW_CALLING_LIST_DBID: 101
                    GSW_CALL_ATTEMPT_GUID: "003DC7H6HG84DBRT1KMIF1TAES000031"
                    GSW_CALL_RESULT: 28
                    GSW_CAMPAIGN_GROUP_DBID: 101
                    GSW_CAMPAIGN_GROUP_DESCRIPTION: ""
                    GSW_CAMPAIGN_GROUP_NAME: "Outbound Campaign Custom@Agent Group Outbound"
                    GSW_CAMPAIGN_NAME: "Outbound Campaign Custom"
                    GSW_CHAIN_ID: 3
                    GSW_CONTACT_MEDIA_TYPE: "voice"
                    GSW_FROM: 0
                    GSW_PHONE: "+33647004"
                    GSW_PHONE_TYPE: 1
                    GSW_RECORD_HANDLE: 284
                    GSW_REFERENCE_ID: 4
                    GSW_SWITCH_DBID: 101
                    GSW_TZ_NAME: "ACT"
                    GSW_TZ_OFFSET: 34200
                    GSW_UNTIL: 86399
                    GSW_USER_EVENT: "ChainedRecord"
                    InteractionSubtype: "OutboundNew"
                    InteractionType: "Outbound"
                }
            ]
        }
    ],
    "userAgent": "WWE Server",
    "protocolVersion": 2
}
```

getRecordFields

<table>
<tr><th>Signature</th><td>getRecordFields(<i>interactionId</i>, <i>succeeded</i>, <i>failed</i>) → {Array.}</td></tr>
<tr><th>Description</th><td>Get the list of outbound fields for an interaction. This method also returns information about whether a field is mandatory and if it can be edited.</td></tr>
<tr><th>Parameters</th><td><table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Signature</th><th>getRecordFields(<i>interactionId</i>, <i>succeeded</i>, <i>failed</i>) → {Array.}</th></tr>
<tr><td></td><td><table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>fails.</td></tr></table></td></tr>
<tr><td><b>Returns</b></td><td>Array.</td></tr>
</table>

Sample request

```
genesys.wwe.service.outbound.getRecordFields('1', succeeded, failed)
```

Sample response

```
{
   "request":"outbound.getRecordFields",
   "data":[
      {
         "name":"GWS_FROM",
         "displayName":"Call From",
         "value":"10.15",
         "isMandatory":true,
         "isEditable":false,
         "type":"time",
         "valueType":"string"
      },
      {
         "name":"GSW_CUSTOM_STRING",
         "value":"Custom message",
         "isMandatory":false,
         "isEditable":true,
         "fieldType":"var-char",
         "valueType":"string"
      },
      {
         "name":"GSW_PHONE_TYPE",
         "displayName":"Phone Type",
         "isEditable":true,
         "isMandatory":false,
         "options":{
            "3":"Business With Extension",
            "2":"Direct Business Phone",
            "10":"Email Address",
            "1":"Home Phone",
            "11":"Instant Messaging",
            "4":"Mobile",
            "7":"Modem",
            "0":"None",
            "6":"Pager",
            "9":"Pin Pager",
            "5":"Vacation Phone",
            "8":"Voice Mail"
         },
         "fieldType":"enum",
         "valueType":"number"
      }
   ],
   "userAgent":"WWE Server",
   "protocolVersion":2
}
```

updateRecordFields

<table>
<tr><th>Signature</th><td>updateRecordFields(<i>interactionId</i>, <i>recordData</i>, <i>succeeded</i>, <i>failed</i>)</td></tr>
<tr><th>Description</th><td>Update one or more outbound fields. The updated fields are sent to Outbound Contact Server when the record is marked done. <b>Note</b>: This operation fails if one of the updated fields does not comply with the data type or mandatory requirements.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>interactionId</td><td>string</td><td>The unique identifier for the interaction.</td></tr>
<tr><td>recordData</td><td>string</td><td>The record data to be updated. This must be an object containing the field names as properties and the values to be updated. The values should comply with the valueType property of the field as returned by getRecordFields. You can update custom fields and the following system fields:
• Call From
• Call Until
• Phone
• Phone Type</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Signature</th><th>updateRecordFields(<i>interactionId</i>, <i>recordData</i>, <i>succeeded</i>, <i>failed</i>)</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>the operation fails.</td></tr>
</table>
</td></tr>
</table>

Sample request

```
genesys.wwe.service.outbound.updateRecordFields(
  '1',
  {
    GSW_FROM: '10.15',
    GSW_UNTIL: '23:45',
    GSW_PHONE_TYPE: 9,
    GSW_CUSTOM_STRING: 'Custom message'
  },
  succeeded,
  failed
)
```

Type definitions

The Outbound namespace includes the following object types:

- Field

Field

<table>
<tr><th>Description</th><td>Represents the JSON structure of a field.</td></tr>
<tr><th>Type</th><td>Object</td></tr>
<tr><th>Properties</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>name</td><td>string</td><td>The name of the field. Use this name in updateRecordFields requests to set or update the value for the field.</td></tr>
<tr><td>displayName</td><td>string</td><td>The name of the field as displayed in Workspace Web Edition. You can use this in a custom view,</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Description</th><th>Represents the JSON structure of a field.</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>if required.</td></tr>
<tr><td>value</td><td>string</td><td>The current value of the field.</td></tr>
<tr><td>isEditable</td><td>boolean</td><td>Specifies whether the field is editable. If updateRecordFields contains a non-editable field, the operation fails.</td></tr>
<tr><td>isMandatory</td><td>boolean</td><td>Specifies whether the field is mandatory. If updateRecordFields tries to set a null or empty value for a mandatory field, the operation fails.</td></tr>
<tr><td>options</td><td>string</td><td>This property is present for fields of type 'enum'. Enums are displayed as dropdowns in Workspace Web Edition. See sample response for getRecordFields for details.</td></tr>
<tr><td>fieldType</td><td>string</td><td>The data type of the field. Possible values are:
• int - Integer
• float - Floating point</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Description</th><th>Represents the JSON structure of a field.</th></tr>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>number
• char - Character
• var-char - String
• date - Date string (MM/DD/YYYY HH:MM)
• time - Time string (HH:MM)
• bool - Boolean
• enum - Key/value pairs</td></tr>
<tr><td>valueType</td><td>string</td><td>The type of value that should be used in updateRecordFields. Possible values are:
• string
• number
• boolean
For example, an enum field may have to be updated with a value type of number. See the sample request for updateRecordFields.</td></tr>
</table>
</td></tr>
</table>

Auth Namespace

Contents

- Developer

Learn about the Auth namespace methods and type definitions in the Service Client API.

> **Important**
>
> Depending on your environment, you might need to contact your Genesys representative to complete the configuration described on this page.

Methods

The Auth namespace includes the following methods:

- getJwtToken

getJwtToken

To use the `auth.getJwtToken` endpoint, you must explicitly define the full endpoint name in the service-client-api-accepted-web-content-origins option. For example: `service-client-api.accepted-web-content-origins = https://genesyspureengage.github.io (*, auth.getJwtToken)`

<table>
<tr><th>Signature</th><td>getJwtToken(<i>succeeded</i>, <i>failed</i>) → {<i>JSON object</i>}</td></tr>
<tr><th>Description</th><td>Get the JWT access token for the current session. If the token is already generated and still valid, it is returned; otherwise a new token is returned.</td></tr>
<tr><th>Parameters</th><td><table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table></td></tr>
<tr><th>Returns</th><td>JSON <code>data</code> object with the token and its expiration date in ISO 8601 date format.
<code>"data": {</code>
<code>   "expiration":</code></td></tr>
</table>

Signature	getJwtToken(*succeeded*, *failed*) → {*JSON object*}

	`"2020-04-14T13:26:51.846Z",`
	`    "jwtToken": ""`
	`}`

Messenger namespace

Contents

- Developer

Learn about the Messenger namespace methods in the Service Client API.

Methods

The Messenger namespace includes the following methods:

- broadcastMessage

broadcastMessage

<table>
<tr><th>Signature</th><td>broadcastMessage(<i>channel</i>, <i>message</i>, <i>succeeded</i>, <i>failed</i>)</td></tr>
<tr><th>Description</th><td>Send a message to other web applications that use the Service Client API and have subscribed to the specified channel.</td></tr>
<tr><th>Parameters</th><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>channel</td><td>string</td><td>The channel to send the message on.</td></tr>
<tr><td>message</td><td>object</td><td>The message (any JSON object) to broadcast on the channel.</td></tr>
<tr><td>succeeded</td><td>function</td><td>A function called when the operation succeeds.</td></tr>
<tr><td>failed</td><td>function</td><td>A function called when the operation fails.</td></tr>
</table>
</td></tr>
</table>

Samples

```
// Add a new message broadcaster:
genesys.wwe.service.messenger.broadcastMessage("my-channel", { foo: "A foo text.", bar: 1234
}, succeeded, failed)
// The operation "broadcastMessage" from the service "messenger" takes a channel name and any
JSON-compliant object.

// In order to receive this message, you must "subscribe" to "my-channel":
```

```
genesys.wwe.service.subscribe([ "messenger:my-channel" ], function(message) {
console.log("message: " + message.data); }, this);
// It is possible to subscribe to several channels:
genesys.wwe.service.subscribe([ "messenger:my-channel", "messenger:my-channel2" ],
function(message) {
  console.log("message: " + message.data + ", channel: " + message.event);
}, this);
```

When a message is broadcast to your channel, you receive an event called `messenger:` with the message in the `data` attribute. For example, here's the event for the broadcast in the sample above:

```
{
  "event": "messenger:my-channel",
  "data": {
    "foo": "A foo text.",
    "bar": 1234
  },
  "userAgent": "WWE Server",
  "protocolVersion": 2
}
```