Workspace Web Edition Developer's Guide and API Reference

Service Client API

12/12/2025

Service Client API

Contents

- 1 Service Client API
  - 1.1 API Overview
  - 1.2 Getting Started
  - 1.3 Security Configuration
  - 1.4 Working with the API

API Overview

You can use the Service Client API to customize how your web application or website integrates with Workspace Web Edition. Genesys provides this API, which is based on window.postMessage, so that your application can access the Workspace Web Edition object model and bypass the cross-domain security limitations.

You can use the Service Client API to perform the following actions:

- Controlling call recording from a third-party application
- Embedding multiple third-party applications in Workspace
- Updating attached data from a third-party application
- Enabling click-to-dial from a third-party application
- Enabling Service Client API to invoke toast in Agent Desktop
- Controlling Case Selection from a Third Party Application

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 Workspace

You can now set the interaction.web-content option to a list of option section names that correspond to web extension views. This means that you can configure 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.

You should also make sure that the service-client-api.accepted-web-content-origins option references all the websites that should use the Service Client API.

See Enabling integration of web applications in the agent interface for details about setting up multiple web applications in Workspace.

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 be sure to configure the service-client-api.user-data.read-allowed and service-client-api.user-data.write-allowed options to enable read and write access to user data.

Enabling Click-to-Dial from a Third-Party Application

If you configure Workspace Web Edition to display your web application in a new tab in the Workspace user interface (as described in Enabling integration of web applications in the agent interface), then the service API only gives access to the dial operation.

Enabling Service Client API to invoke toast in Agent Desktop

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.

Getting Started

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

1. You have a web application that you've integrated in Workspace Web Edition. See Enabling integration of web applications in the agent interface for details.
2. Download the sample application: service-client-api.zip.

3. Copy the **wwe-service-client-api.js** file in the sample application to a location your web application can access.
4. Set the options described below in 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
    - Email Namespace
    - Interaction Namespace
    - Media Namespace
    - System Namespace
    - Voice Namespace

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 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
- media.getMediaList
- media.setState
- voice.dial
- 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.<service-name>.

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 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 `<script>` 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:

```
<html>
    <head>
        <script src="wwe-service-client-api.js"></script>
        <script>

            function test() {
                genesys.wwe.service.sendMessage({
                    request: "agent.get"
                }, function(result) {
                    console.debug("SUCCEEDED, result: " + JSON.stringify(result, null, '\t'));
                }, function(result) {
                    console.debug("FAILED, result: " + JSON.stringify(result, null, '\t'));
                });
```

```
            }

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

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

        </script>
    </head>
    <body>
        Hello world
    </body>
</html>
```

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.<Service name>.<Service function>(<... function parameters ...>,
[<optional done() callback>, [<optional fail() callback>]]);
```

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

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.

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.
NEW_MESSAGE	This event represents a new message.
ERROR	An error occurs in the interaction.
Voice events	
CALL_RECORDING_STATE_CHANGED	The call recording state changed.
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	

eventType	Description
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)	
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.