Service Client API Reference

Service Client API

1/17/2022

Contents

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

> **Important**
> You must contact your Genesys representative to configure Agent Desktop to use the Service Client API.

Use the Service Client API to customize how your web application or website integrates with Agent Desktop. This JavaScript API is based on window.postMessage and provides methods your application can use to communicate cross domain with Agent Desktop 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 Desktop—contact your Genesys representative to enable integration of web applications in Agent Desktop.
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—contact your Genesys representative to provision this 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 **Note:** You must work with your Genesys representative to enable and use this part of the Service Client API.
    - Email Namespace
    - Interaction Namespace
    - Media Namespace
    - System Namespace
    - Voice Namespace

- Outbound Namespace
- Auth Namespace

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

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

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.
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.
Voice events	

eventType	Description
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	
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.

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

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

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

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 Desktop

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 Agent Desktop 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.

Contact your Genesys representative to enable embedding multiple third-party applications in Agent Desktop.

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 options related to user data in the Service Client section of Agent Setup to enable read and write access to user data.

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

If you configure Agent Desktop to display your web application in a new tab in the Agent Desktop user 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.

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
}
```