Cisco Finesse Web Services Developer and JavaScript Guide, Release 12.6(1)

First Published: 2020-04-24

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

CHAPTER 1

Introduction

What's New in Cisco Finesse 12.6(1)

REST APIs

The following APIs have been added in Cisco Finesse 12.6(1).

- Device—Get List of Devices for Extension—This API allows a user to retrieve the list of devices associated with an extension. This API is only supported for Unified CCE deployments.
- Finesse MaintenanceMode—This API allows the user to request Finesse to move to maintenance mode. The following are the new Finesse MaintenanceMode APIs:
  - Finesse MaintenanceMode—Get
  - Finesse MaintenanceMode—Update
- ConnectedUserInfo—This API retrieves the list of agent details logged in during the real time. The following are the new ConnectedUserInfo APIs:
  - ConnectedUserInfo—Summary
  - ConnectedUserInfo—Get Connected Users Information

The following changes are made to the payloads in Cisco Finesse 12.6(1) REST APIs:

- User APIs—The following fields are added to the payload:
  - deviceSelection—Indicates whether the CTI device selection is enabled for the agent.
  - activeDeviceId—A unique ID of the active device associated with the extension to which the agent is signed in.

  - Devices—Information about the list of devices associated with an extension.
  - skillTargetId—Indicates the unique identifier for the skill target assigned to the agent in the Unified CCE database.
  - services—Information about the list of services that are configured for an agent or a supervisor.

- Dialog—Drop Participant from Conference—This API allows an agent or supervisor to make a request to drop other participants from a conference based on the permission set by the administrator.
  - services—Information about the list of services that are associated with a dialog.
- Single Sign-On APIs—The optional parameters are added in the Fetch Access Token and Refresh Existing Access Token APIs
- SystemInfo APIs—The following fields are added to the payload:
  - ctiTimeInMMode—The total time (in seconds) that the CTI server is in maintenance mode.
  - ctiMMode—Indicates whether the CTI server is in maintenance mode.
  - ctiServers—Information about the list of CTI servers that the Cisco Finesse is connected to.
  - finesseTimeInMMode—The total time (in seconds) that the Finesse server is in maintenance mode.
  - finesseMMode—Indicates whether the Finesse server is in maintenance mode.

Finesse Desktop Gadget Development

A new subsection is added for Multi-Tab gadgets functionality.

jQuery

The jQuery version hosted by Finesse has been upgraded from 3.4.1 to 3.5.1.

JavaScript APIs

The `finesse.containerservices.TaskActivityNotification` JavaScript APIs has been added in Cisco Finesse 12.6(1).

The following changes are made to the payloads in Cisco Finesse 12.6(1) JavaScript APIs:

- Gadget Configuration—Added the skillTargetId field which refers to the skill ID of the user.
- User—The following functions are added:
  - getActiveDeviceId()—Retrieves the current active device ID of the agent.
  - getDevices()—Retrieves the list of devices associated with a particular extension.
  - getSkillTargetId—Retrieves the Id for the skill target assigned to the user in the Unified CCE database.
  - getServices()—Retrieves the list of services configured for the user.
  - isDeviceSelectionEnabled—Retrieves whether the device selection is enabled for the user.

    - loginWithActiveDeviceId()—Performs an agent login for a user and associates the agent with the specified extension and device.

- Container Services—The following services are added:
    - enableTitleBar()—Displays the title bar for a page-level gadget which is not visible by default.
    - getMyGadgetView()—Returns the current view details of the gadget.
    - hideCertificateBanner()—Request to hide the Certificate Banner.
    - hideMyGadget()—Makes the current gadget inaccessible by hiding it from the title bar of the multi-tab gadget.
    - hideMyGadgetNotification()—Removes the current gadget's notifications from the title bar of the multi-tab gadget.
    - isTabbedGadget()—Checks if the gadget is configured inside a multi-tab gadget.
    - setMyGadgetTitle(title)—Sets the title of the current gadget.
    - showCertificateBanner()—Displays the Certificate Banner with message "Gadget certificates are yet to be accepted."
    - showMyGadget()—Makes the current gadget accessible by showing its tab in the title bar of the multi-tab gadget.
    - showMyGadgetNotification()—Makes a red dot appear on the gadget title if it is not the current active gadget. The notification dot disappears if the tab is active.
- Container Services Topics—The following service is added:
    - FINESSE_MAINTENANCE_MODE_EVENT—Listens to notification related to Finesse maintenance mode changes.
- SystemInfo—The following functions are added:
    - getCtiMMode—Retrieves the CTI server in maintenance mode.
    - getCtiTimeInMMode—Retrieves the total time (in seconds) that the CTI server is in maintenance mode.
    - getCtiServers—Retrieves the list of CTI servers that Cisco Finesse is connected to.
    - getFinesseMMode—Retrieves the Finesse server in maintenance mode.
    - getFinesseTimeInMMode—Retrieves the total time (in seconds) that the Finesse server is in maintenance mode.

Deprecated Features

Notifications over BOSH (Long Polling)

In this release, support for notifications over BOSH (long polling) is deprecated. Applications that require notifications are recommended to use WebSocket-based notifications (Finesse desktop) or notifications over direct XMPP (over TCP).

The usage of port 7443 is deprecated and the port 8445 should be used instead. For the details on how to use port 8445 for WebSocket notifications, refer to the *Managing Notifications in Third-Party Applications* section of the *Cisco Finesse Web Services Developer and JavaScript Guide*.

Cisco Finesse REST APIs

This document is the official reference for the Cisco Finesse Application Programming Interface (API). The Finesse desktop APIs support the Finesse desktop, providing agent desktop functionality, such as call control and state changes.

The Finesse configuration APIs support the Finesse administration console, providing the ability to configure resources (such as reason codes, wrap-up reasons, and workflows).

The Finesse APIs support the following capabilities:

- User Sign In/Sign Out
- Agent States
- Configurations
- Subscriptions
- Call Control
- Reason Codes
- Wrap-up Reasons
- Teams
- Team Resource
- Queues
- Task Routing
- Mobile Agents
- Workflows
- TeamMessages
- Desktop Chat

This guide explains each API and the notification messages returned by the APIs. The guide includes a section to assist developers with running and validating the APIs in a lab environment.

REST API Response Caching

The Finesse webproxy caches the following REST API responses:

- `ChatConfig`
- `ECCVariableConfig` (applicable for Unified CCE)
- `MediaDomain` (applicable for Unified CCE)
- `TeamResource`: The responses of the TeamResource API are cached at the team-level.

  When Reason Codes, Wrap-Up Reasons, Phone Books, Workflows, or Media Properties Layouts specific to a Team are updated, the Finesse webproxy cache is cleared and the update is reflected in the next TeamResource API request.

Proxy cache bypassing degrades performance and is only recommended for debugging purposes during the gadget development or troubleshooting.

- To bypass the server cache for the Finesse API, include `bypassServerCache=true` as a query parameter in the request or clear server cache using the CLI **utils webproxy cache clear rest**.
- To bypass the server cache for the Finesse desktop, include `bypassServerCache=true&nocache` as a query parameter in the desktop URL.

For more information on the CLI commands, see *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

JavaScript Library and Sample Gadgets

Finesse provides a JavaScript library (finesse.min.js) and several sample gadgets to help jump-start your gadget development. The JavaScript library provides a substantial amount of fundamental code infrastructure that you would otherwise must write yourself.

The Cisco Finesse JavaScript library is dependent on the jQuery library, which needs to be manually imported in the gadget code. Starting Finesse 12.6(1), the hosted jQuery version has been upgraded to 3.5.1. Gadgets that use jQuery directly should evaluate if there are any impacts due to the upgrade. The jQuery Migrate development tool can help resolve any upgrade issues. For more information about the jQuery Migrate development tool, see https://github.com/jquery/jquery-migrate/.

- You can access the JavaScript library at the following URL: https://<FQDN>:<port>/desktop/assets/js/finesse.min.js

Note The unminified version of finesse.min.js can be accessed from the URL: `https://<FQDN>:<port>/desktop/assets/js/finesse.js`

- For JavaScript API documentation, refer to the Cisco Finesse JavaScript APIs chapter.
- You can access JQuery at the following URL: https://<FQDN>:<port>/desktop/assets/js/jquery.min.js.
- If you have third-party gadgets that are loaded on Finesse, the third-party gadgets can access the JavaScript library at: /desktop/assets/js/finesse.min.js.

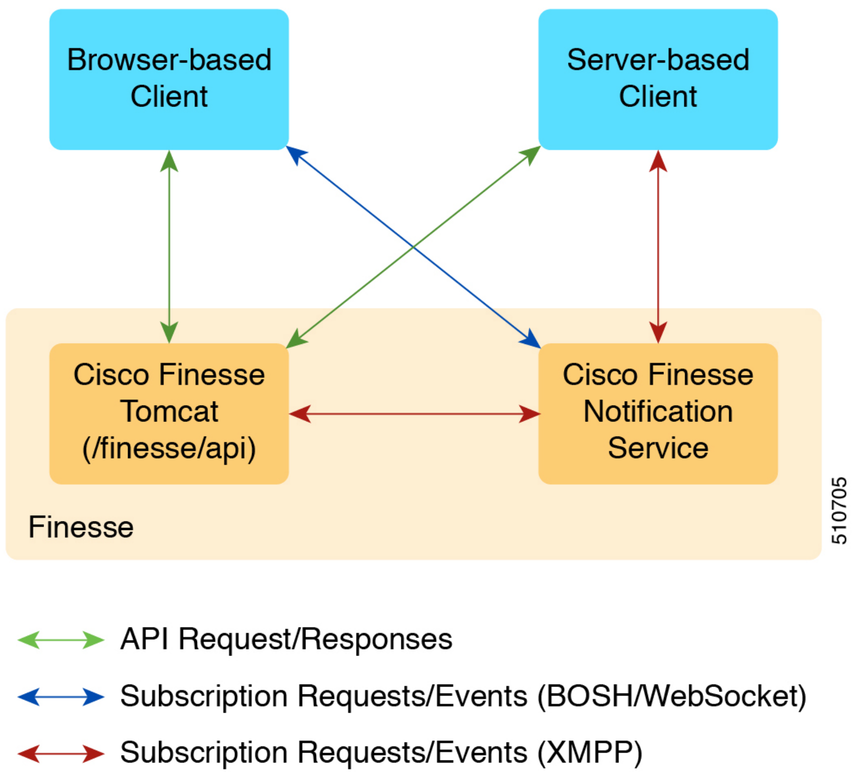
- The sample gadgets are available from Cisco DevNet at the following link: https://developer.cisco.com/site/finesse/.

Note For the proper functioning of the JavaScript library, you must import both the Finesse JavaScript library and JQuery.

Communication with the Cisco Finesse Web Service

The Cisco Finesse Notification Service name in the following diagram is specific to Unified CCE deployments. In a Unified CCX deployment, the notification service is named the Cisco Unified CCX Notification Service.

Figure 1: Finesse API and Event Flow


Browser-based Client
Server-based Client
Cisco Finesse Tomcat (/finesse/api)
Cisco Finesse Notification Service
Finesse
510705
API Request/Responses
Subscription Requests/Events (BOSH/WebSocket)
Subscription Requests/Events (XMPP)

Note The Finesse desktop supports receiving updates through BOSH/WebSocket only.

Client Requests

Cisco Finesse Release 12.5(1) or higher supports only secure HTTP (HTTPS) requests from clients. Cisco Finesse desktop operations can be performed using the available REST HTTPS request described in this guide.

Operations on specific objects are performed using the ID of the object in the REST URL. For example, the URL to view a single object (HTTPs) would be:

The URL to view a single object (HTTPS) would be:

```
https://<FQDN>:<port>/finesse/api/<object>/<objectID>
```

FQDN is the fully-qualified domain name of the Finesse server.

Finesse configuration APIs require the application user ID and password, which is established during installation, for authentication purposes.

Finesse APIs use the following HTTP methods to make requests:

- GET: Retrieve a single object or list of objects (for example, a single user or list of users).
- PUT: Replace a value in an object (for example, to change the state of a user from NOT_READY to READY).
- POST: Create a new entry in a collection (for example, to create a new reason code or wrap-up reason).
- DELETE: Remove an entry from a collection (for example, to delete a reason code or wrap-up reason).

Finesse uses the standard HTTP status codes (for example, 200, 400, and 500) in the response. These status codes indicate overall success or failure of the request.

If an API operation fails, a detailed error is returned in the HTTP response message body. The error, in XML format, appears as follows:

```
<ApiErrors>
   <ApiError>
       <ErrorType>type</ErrorType>
       <ErrorMessage>message</ErrorMessage>
       <ErrorData>data</ErrorData>
   </ApiError>
</ApiErrors>
```

Finesse has a Dependency Manager that collects the state of internal dependencies for Finesse (such as the state of the Cisco Finesse Notification Service) and reports these states to external entities.

If any of these dependencies are down, Finesse is out of service. If the Cisco Finesse Tomcat is running, Finesse rejects any API requests and returns an HTTP 503 error. The error appears as follows:

```
<ApiErrors>
  <ApiError>
    <ErrorType>Service Unavailable</ErrorType>
    <ErrorData></ErrorData>
    <ErrorMessage>SERVER_OUT_OF_SERVICE</ErrorMessage>
  </ApiError>
</ApiErrors>
```

If the Cisco Finesse Tomcat service is not running, Finesse returns a Connection Timeout error.

All Finesse APIs use HTTP BASIC authentication, which requires the credentials to be sent in the "Authorization" header. The credentials contain the username and password, separated by a single colon (:), within a BASE64-encoded string. For example, the Authorization header would contain the following string:

```
"Basic YWdlbnRiYXJ0b3dza2k6Y2FybWljaGFlbA=="
```

where "YWdlbnRiYXJ0b3dza2k6Y2FybWljaGFlbA==" is the Base64-encoded string of "agentbartowski:carmichael" (agentbartowski being the username and carmichael being the password).

In case of Single Sign-On mode, the Authorization header would contain the following string:

```
Bearer <authtoken>
```

where the authtoken has to be fetched from IDS through the ADFS server.

If an administrator changes the password for an agent or supervisor on the secondary Administration & Data server (if configured) while the primary distributor process on Unified CCE is down, the agent or supervisor can still use the old password and access all REST APIs except the sign-in request. To ensure this does not happen, the primary distributor must be up and running when the administrator changes the password.

HTTPS Requests

Cisco Finesse does not support plain HTTP but supports only secure HTTP (HTTPS). In response to clients accessing Finesse using plain HTTP, the 301 HTTP redirect is issued to the secured port 8445.

Note Cisco Finesse supports HTTP/2 protocol by default.

Clients must make all HTTPS requests to port 8445. Finesse desktop APIs conform to the following format:

```
https://<FQDN>:<port>/finesse/api/<object>
```

Note Use the fully qualified domain name (FQDN) of the Finesse server instead of the IP address to avoid address mismatch errors (SSL certificate uses the Finesse hostname.)

The following ports are disabled by default:

- BOSH/WebSocket (HTTP)—7071
- XMPP—5222

Use the CLI command **utils finesse set_property webservices enableInsecureOpenfirePort true** to enable these ports. For more information on CLI commands, see *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

Note For gadget development, Finesse server and client connections only support TLS 1.2 by default.

Real-Time Events

Real-time events (such as call events, state events, and so on) are sent by the Cisco Finesse Notification Service, using the XEP-0060 Publish-Subscribe extension of the XMPP (Extensible Messaging and Presence Protocol) protocol. Applications that need to communicate with the Notification Service must use XMPP over the BOSH (Bidirectional-streams Over Synchronous HTTP)/WebSocket transport.

All real-time events are sent over HTTPS.

BOSH/WebSocket is an open technology for real-time communication and is useful for emulating a long-lived, bidirectional TCP connection between two entities (such as client and server). See documentation at the XMPP Standards Foundation (http://www.xmpp.org) for details about both XMPP and BOSH/WebSocket (XEP-0124).

Client applications can communicate with the Cisco Finesse Notification Service through BOSH/WebSocket over HTTPS, using the binding URI https://<FQDN>:7443/http-bind. Developers can create their own BOSH/WebSocket library or use any that are available publicly.

After creating the connection, applications can receive notification events of feeds to which they are subscribed. Users are currently subscribed to a few feeds by default (subject to change). Other feeds require an explicit subscription (see *Subscription Management*).

Note The agents must be connected to the Cisco Notification Service to retain the user presence information. If not, it can result in unexpected behavior or cause the desktop not to respond as expected.

API Parameter Types

The following sections describe the parameter and data types for the Cisco Finesse Desktop Interface APIs.

API Header Parameters

Name	Type	Description
password	String	The password used in the request header to make any Finesse API request. Finesse supports a "Basic" authorization scheme only and authorization is required for each Finesse API request.
username	String	The username used in the request header to make any Finesse API request. Finesse supports a "Basic" authorization scheme only and authorization is required for each Finesse API request.

Path Parameter

A path parameter is included in the path of the URI. In the following example, *dialogId* is a path parameter.

```
https://<FQDN>:<port>/finesse/api/Dialog/<dialogId>
```

Query Parameter

A query parameter is passed in a query string on the end of the URI you are calling. The query parameter is preceded by a question mark. Multiple query parameters are connected by an ampersand (&). In the following example, *category* is a query parameter.

```
https://<FQDN>:<port>/finesse/api/User/<id>/ReasonCodes?category=NOT_READY
```

Data Types

The following table lists the data types used in API parameters and notification message fields.

Type	Description
Boolean	A logical data type that has one of two values: true or false.
Integer	A 32-bit wide integer.
Long	A 64-bit wide integer.
String	A variable-length string. If a maximum length exists, it is listed with the parameter description.

Cisco Finesse API Errors

Error codes for Cisco Finesse are categorized as follows:

- 4xx—Client-related error
- 5xx—Server-related error

Each error includes a failure response, error type, error message, and error data. The following is an example of a failure message format:

```
<ApiErrors>
   <ApiError>
      <ErrorType>Authentication Failure</ErrorType>
      <ErrorMessage>UNAUTHORIZED</ErrorMessage>
      <ErrorData>jsmith</ErrorData>
   </ApiError>
</ApiErrors>
```

In addition to Cisco Finesse API errors, a response may return a CTI error or an HTTP error.

Note This document contains information about error type and error message. You can find information about error data values for most User and Dialog errors in the following documents:

For Finesse deployments with Unified CCE, see the *CTI Server Message Reference Guide for Cisco Unified Contact Center Enterprise*, which you can find at https://developer.cisco.com/site/cti-protocol/documentation/ .

For Finesse deployments with Unified CCX, see the https://developer.cisco.com/docs/contact-center-express/#!cti-protocol-dev-guide.

HTTP Errors

All HTTP errors are returned as HTTP 1.1 Status Codes. Errors that might be for Finesse-specific events are listed below:

500 Internal Server Error
Finesse Web Services returns 500 if the CTI connection is lost but the loss is not yet detected by automated means.

- 500 - DB_RUNTIME_EXCEPTION (database error, but the database is thought to be operational)
- 500 - RUNTIME_EXCEPTION (a non-database error)
- 500 - AWS_SERVICE_UNAVAILABLE (AWS not operational)

503 Service Unavailable

If Finesse is in PARTIAL_SERVICE or OUT_OF_SERVICE, it returns 503 for all requests. If any dependent service goes down, Finesse goes to OUT_OF_SERVICE state (for example, if the Cisco Finesse Notification Service is down).This error is due to a temporary outage or overloading condition. A retry after several seconds is likely to succeed. For example, the system returns 503 when the system is just starting up and when the system is trying to connect to the CTI server.

Peripheral Error Codes

Cisco Finesse, Release 12.5(1) introduces peripheral error codes for CTI operations, which provide a more detailed description of the error scenario. The newly added parameters are:

- peripheralErrorCode
- peripheralErrorMsg
- peripheralErrorText

Example:

```
<ApiErrors>
    <ApiError>
        <ErrorType>Service Unavailable</ErrorType>
        <ErrorData></ErrorData>
        <ErrorMessage>SERVER_OUT_OF_SERVICE</ErrorMessage>
        <peripheralErrorCode>13036</peripheralErrorCode>

<peripheralErrorMsg>PERERR_GW_E_JTAPIOBJ_PERFORMANSWERCALL_NO_TERMINAL_CONNECTION</peripheralErrorMsg>

        <peripheralErrorText>The routine performAnswerCall in class JTapiObj got a null
connection from a call to 'findTerminalConnection'</peripheralErrorText>
    </ApiError>
</ApiErrors>
```

For more information, see *Cisco IPCC Error Codes* at https://www.cisco.com/c/en/us/support/docs/voice-unified-communications/unified-contact-center-enterprise/26142-error-codes.html.

CHAPTER 2

Lab Development Environment Validation with Cisco FinesseWeb Services APIs

Environment and Tools

The topics in this section are for use as a learning exercise and are not meant for use in real deployments.

To complete these exercises, you need the following:

- A user who is configured as an agent in Unified CCE or Unified CCX (with an agent ID, password, and extension). Make the agent a member of a team and of a queue. (A queue is a skill group.)
- Three phones that are configured in Cisco Unified Communications Manager: one for the agent, one for the caller, and one to use for conferencing and transfer APIs. These can be Cisco IP "hard phones" or Cisco IP Communicator softphones.
- Tools: Postman and Pidgin for Windows or Adium for Mac OS X.

Note Postman, Pidgin and Adium are meant to aid in development; however, they are not officially supported.

Postman

Procedure

Postman is an example of a REST client utility that allows you to send HTTP requests to a specific URL. You can use this utility in your lab to exercise the Finesse Web Service APIs by entering the URI for an API and checking the response. All APIs are accessible by URI and follow a request/response paradigm. There is always a single response for any request.

You can download Postman from https://www.getpostman.com/.

For using self-signed SSL certificates with Postman see, http://blog.getpostman.com/2014/01/28/using-self-signed-certificates-with-postman/

To test an API in Postman, follow these steps:

Step 1 Copy and paste the URI for the API request from this Developer Guide into a text editor. For example, to enter the URI for signing in, copy the URI from the *User—Sign In to Finesse* API. Examine the pasted code for case sensitivity and format and remove any carriage returns.

Step 2 Update the URI with the IP address of your Cisco Finesse Web Services server.

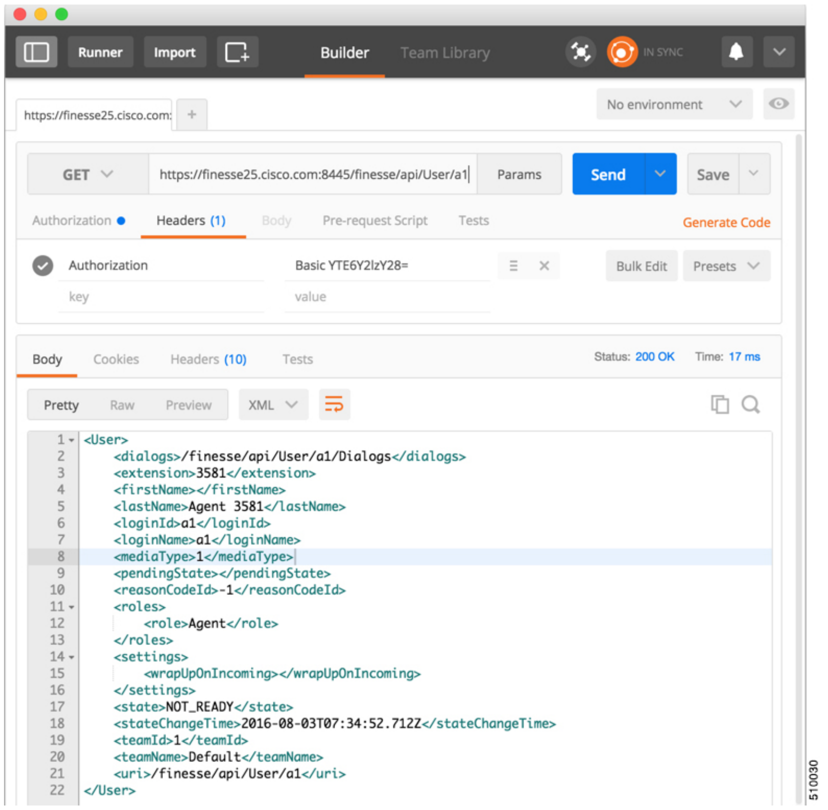
Step 3 Add any mandatory parameters for the request.

Step 4 Enter the username and password for the agent you set up for these exercises.

Step 5 For Content Type, enter **application/xml**

Step 6 Click the appropriate action (GET, PUT, or POST).

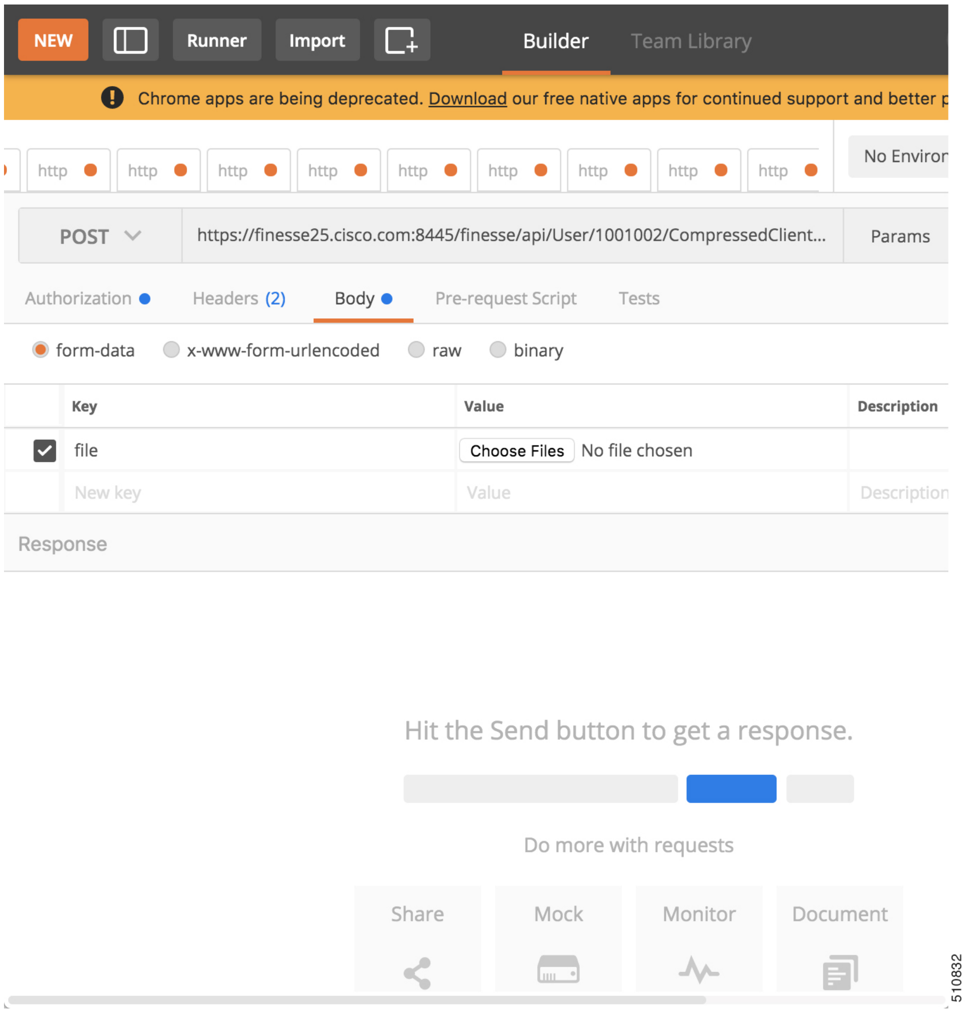
Figure 2: Postman Rest Client



```
<User>
    <dialogs>/finesse/api/User/a1/Dialogs</dialogs>
    <extension>3581</extension>
    <firstName></firstName>
    <lastName>Agent 3581</lastName>
    <loginId>a1</loginId>
    <loginName>a1</loginName>
    <mediaType>1</mediaType>
    <pendingState></pendingState>
    <reasonCodeId>-1</reasonCodeId>
    <roles>
        <role>Agent</role>
    </roles>
    <settings>
        <wrapUpOnIncoming></wrapUpOnIncoming>
    </settings>
    <state>NOT_READY</state>
    <stateChangeTime>2016-08-03T07:34:52.712Z</stateChangeTime>
    <teamId>1</teamId>
    <teamName>Default</teamName>
    <uri>/finesse/api/User/a1</uri>
</User>
```

When you send zip files, select Content Type as form-data. For more information, see CompressedClientLog—Post Compressed Log to Finesse, on page 188.

Figure 3: Send Zip File



Chrome apps are being deprecated. Download our free native apps for continued support and better p

POST https://finesse25.cisco.com:8445/finesse/api/User/1001002/CompressedClient...

Body: [✓ form-data] x-www-form-urlencoded raw binary

	Key	Value	Description
✓	file	Choose Files No file chosen	
	New key	Value	Description

Pidgin for Windows

Pidgin is a multiplatform instant messaging client that supports many common messaging protocols, including XMPP. You can use Pidgin to establish an XMPP connection and view XMPP messages published by the Cisco Finesse Notification Service.

Note You cannot be signed in to Pidgin at the same time you are signed in to Finesse as the XMPP event feed is disrupted.

Notifications that result from API requests made in Postman appear in the XMPP Console tool of the Pidgin application. For example, if you use Postman to change an agent's state, you can see the resulting agent state change event in the Pidgin XMPP Console window.

Note Make sure that you use the same username and resource values in both Postman and Pidgin.

You can download Pidgin from http://www.pidgin.im/download/.

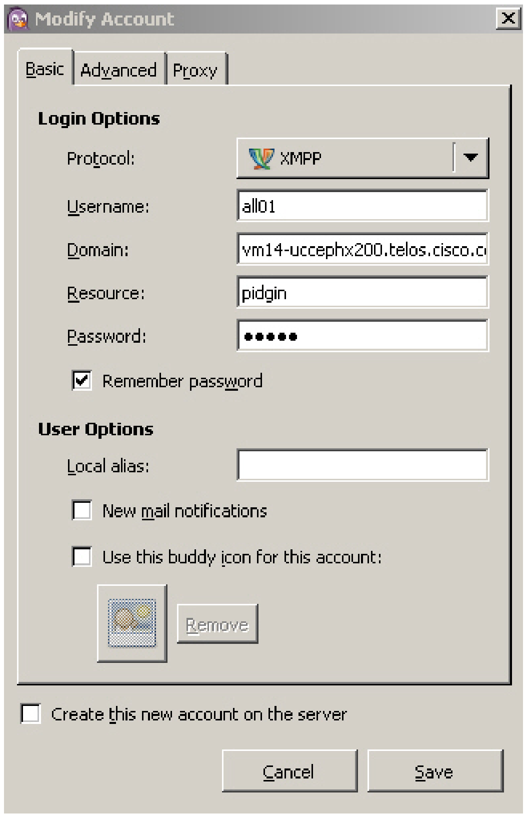
Perform the following steps to configure XMPP:

1. In Pidgin, go to **Tools** > **Plugins** to open the Plugins dialog box.
2. Check the **XMPP Console** and **XMPP Service Discovery** check boxes.

Perform the following steps to configure Pidgin:

1. Add an account for your XMPP server. Go to **Pidgin** > **Accounts** > **Manage Accounts** > **Add Account**. The Add Account dialog box opens.
2. For Protocol, select **XMPP**.
3. For Username, enter the username for the agent that you added.
4. For Domain, enter the fully-qualified domain name of the Cisco Finesse server.
5. For Resource, enter any text.
6. For Password, enter the password of the agent.

Figure 4: The Pidgin Interface



Modify Account

Login Options
Protocol: XMPP
Username: all01
Domain: vm14-uccephx200.telos.cisco.c
Resource: pidgin
Password: •••••
[✓] Remember password

User Options
Local alias:
[ ] New mail notifications
[ ] Use this buddy icon for this account:

[ ] Create this new account on the server

7. Click **Save**.
8. Click the **Advanced** tab.
9. Check the **Allow plaintext auth over unencrypted streams** check box.

10. For Connect Server, enter the IP address of the Finesse server.
11. If the Connection Security drop-down menu is present, choose **Use encryption if available**.
12. Click **Save**.

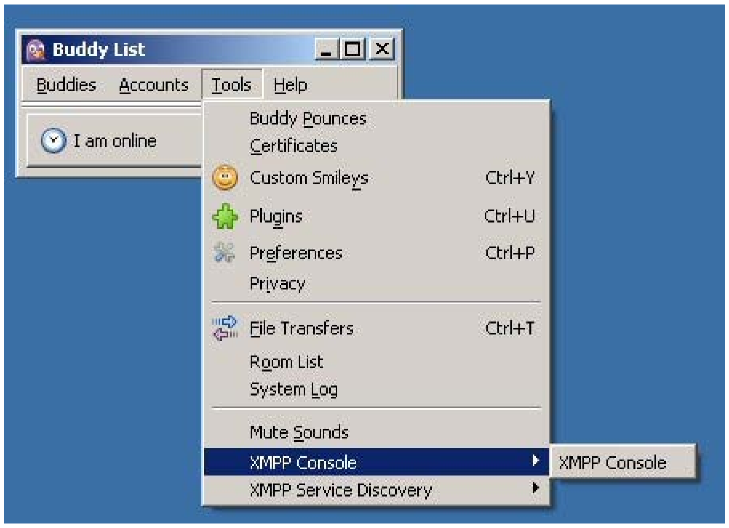
Note Connect port and File transfer proxies should be filled in automatically (5222 should appear in the Connect port field).

Note When connecting to the secure port 5223:

1. Add the Finesse Notification Service certificate in the Pidgin certificate manager. Finesse Notification Service shares the same certificate with Cisco Finesse Tomcat.
2. To download the certificate:
    a. Sign in to the Cisco Unified Operating System Administration through the URL (https://FQDN:8443/cmplatform, where FQDN is the fully qualified domain name of the primary Finesse server and 8443 is the port number).
    b. Click **Security** > **Certificate Management**.
    c. Click **Find** to get the list of all the certificates.
    d. In the Certificate List screen, choose **Certificate** from the **Find Certificate List where** drop-down menu, enter **tomcat** in the **begins with** option and click **Find**.
    e. Click the FQDN link which appears in the **Common Name** column parallel to the listed tomcat certificate.
    f. In the pop-up that appears, click the option **Download .PEM File** to save the file on your desktop.
3. In the Pidgin Certificate Manager, go to the Connection Security drop-down menu and choose **Use old-style SSL**.
4. In the **Connect Server** field, enter the FQDN of the Finesse server.

The XMPP logo next to the agent's name becomes active (is no longer dimmed). To see event messages in Pidgin, open the XMPP Console.

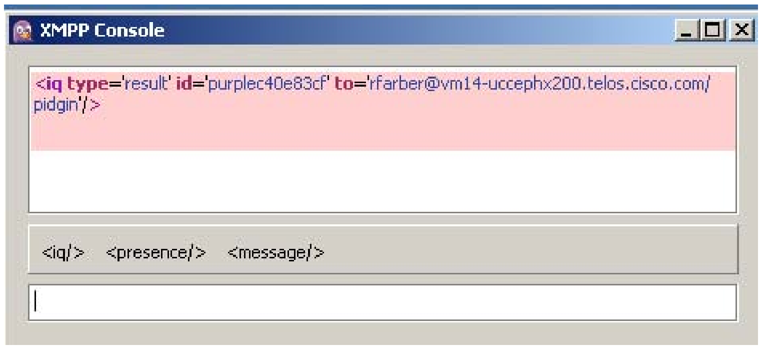
Figure 5: Open XMPP Console in Pidgin



Note The agent must be signed in to Finesse through Postman or the browser interface to be signed in to the XMPP account on Pidgin.

The XMPP Console window immediately begins to update every few seconds with iq type statements. The window does not display an event message until an event occurs. If the XMPP Console window fills with iq type notifications and becomes difficult to navigate, close and reopen it to refresh with a clean window.

Figure 6: The XMPP Console Window



```
<iq type='result' id='purplec40e83cf' to='rfarber@vm14-uccephx200.telos.cisco.com/
pidgin'/>
```

Adium for Mac OS X

Adium is a free open source instant messaging application for Mac OS X. You can use Adium to establish an XMPP connection and view XMPP messages published by the Cisco Finesse Notification Service.

You can download Adium from https://www.adium.im.

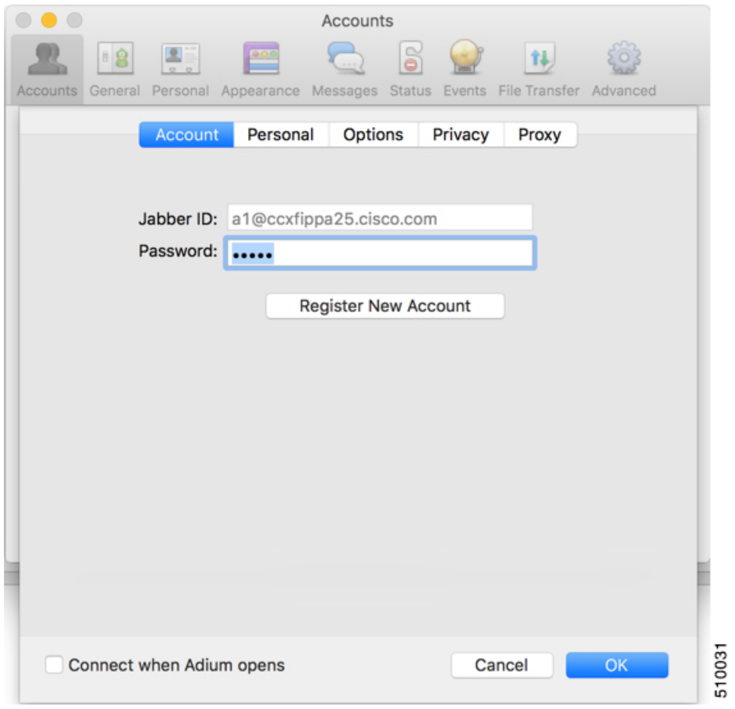
Perform the following steps to configure XMPP:

1. In Adium go to **Preferences** > **Account** > **'+'** > **XMPP (Jabber)**.

2. For Jabber ID, enter the username for the agent along with the fully qualified domain name of the Cisco Finesse server.

3. For Password, enter the password of the agent.

Figure 7: The Adium Interface



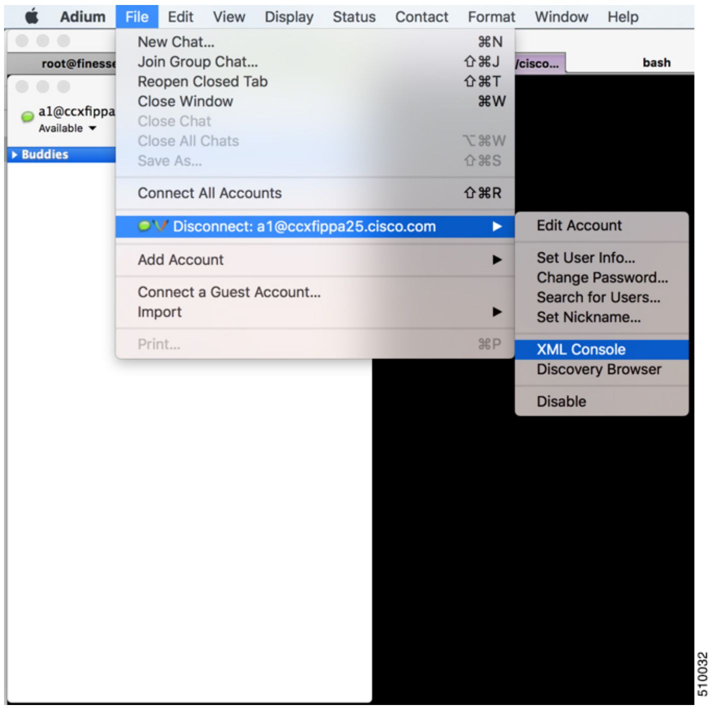
Jabber ID: a1@ccxfippa25.cisco.com
Password: •••••
☐ Connect when Adium opens

4. Enable XMPP Advanced Features (Default: Off).

   To enable the XML Console menu run the following command in Terminal: `defaults write com.adiumX.adiumX AMXMPPShowAdvanced -bool YES`

5. In Adium go to **File** > **Logged in User** > **XML Console**.

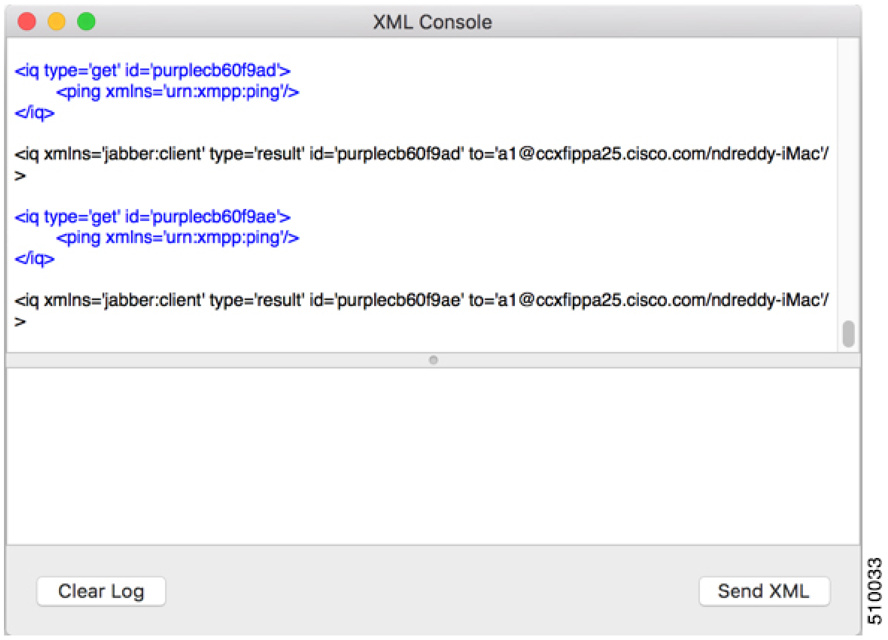
Figure 8: Open XML Console in Adium



Note The agent must be signed in to Finesse through Postman or the browser interface to be signed in to the XMPP account on Adium.

The XML Console window immediately begins to update every few seconds with iq type statements. The window does not display an event message until an event occurs. If the XML Console window fills with iq type notifications and becomes difficult to navigate, close and reopen it to refresh with a clean window.

Figure 9: The XML Console Window



```
<iq type='get' id='purplecb60f9ad'>
      <ping xmlns='urn:xmpp:ping'/>
</iq>

<iq xmlns='jabber:client' type='result' id='purplecb60f9ad' to='a1@ccxfippa25.cisco.com/ndreddy-iMac'/
>

<iq type='get' id='purplecb60f9ae'>
      <ping xmlns='urn:xmpp:ping'/>
</iq>

<iq xmlns='jabber:client' type='result' id='purplecb60f9ae' to='a1@ccxfippa25.cisco.com/ndreddy-iMac'/
>
```

Cisco Finesse APIs

APIs that control actions on the Finesse desktop and call control make use of two objects:

- User object: The User object represents agent and supervisor data and actions. This object is used to get information about a single user or list of users, to sign in or out of the Finesse Desktop, and change agent state.
- Dialog object: The Dialog object represents a dialog with participants. For media type "voice", this object represents a call. A participant can represent an internal user (such as an agent) or an external user (for example, a customer). A participant can belong to only one dialog but a user can be a participant in several dialogs. The Dialog object is used for call control and call data.

GET requests are synchronous. That is, the response body of a successful GET request contains all requested contents, which you can view in Postman or RESTClient. No event is published by XMPP and no event is received in Pidgin.

PUT and POST requests are asynchronous. A successful response is an HTTP return code of 200 or 202. The response body does not contain the updated object information.

If a PUT, POST, or DELETE request is on a User or Dialog object, the update is published by XMPP as a real-time event to Pidgin. If a PUT, POST, or DELETE request is on a configuration object (for example, a ReasonCode object), XMPP does not publish a real-time update. You must perform a GET request to get an updated copy of the object.

GET, PUT, POST, and DELETE requests that fail Finesse server internal checks are synchronous. If a request fails, Postman or RESTClient display the error. No event is published by XMPP to Pidgin. However, if the request fails on CTI side, Finesse will send an api Error XMPP event back to client after receiving a failure confirmation response from the CTI Server.

For each object, Finesse maintains an internal request queue where each subsequent request for this object is processed only after a success or a failure confirmation response is received from the CTI Server for the previous request.

RequestId is a user provided unique string that is added to the request API header and used to correlate originating requests with the resulting XMPP notifications or errors.

Note RequestId is a best effort request-response correlation and is not reliable.

XMPP event notifications that match the requested action are tagged with the requestId (if available) from the original request. If the originating request results in a system error, the corresponding XMPP error notifications also contain the requestId. Note that the request id is not sent in the case of synchronous responses to GET requests. Although not mandatory, using a unique requestId helps in tracking error messages and allows a user to debug issues faster, as messages with requestId are easily tracked in Finesse logs.

Note The requestId facility is not implemented for Task routing APIs. For more information, see the section on *Task Routing APIs.*

The following sections provide instructions and examples for using the APIs with Postman and Pidgin.

Sign In to Finesse

Use the User - Sign In to Finesse API to sign the agent in.

This example uses the following information:

- Finesse server FQDN: finesse1.xyz.com
- Agent name: John Smith
- Agent ID: 1234
- Agent password: 1001
- Agent extension: 1001
- requestId: xyz

Note This example shows the URL field for a Unified CCE deployment. In a Unified CCX deployment, you must include the port number in the URL.

1. Access Postman (Ctrl + Alt +P from the Mozilla Firefox browser) and enter the following string in the URL field:

   ```
   https://finesse1.xyz.com/finesse/api/User/1234
   ```

2. Enter the agent's ID (1234) and password (1001) in the two User Auth fields directly under the URL field.
3. In the Content Type field, enter application/XML.

4. In the area under Content Options, enter the following:

```
<User>
 <state>LOGIN</state>
 <extension>1001</extension>
</User>
```

5. (Optional) To add the requestId:
    a. Click **Headers**.
    b. In the Name field, enter **`requestId`**, and in the Value field, enter **`xyz`**.
    c. Click **Add/Change**

6. Click **PUT**.

   Postman returns the following response:

```
PUT on https://finesse1.xyz.com/finesse/api/User/1234
Status 202: Accepted
```

   Finesse returns a user notification, which you can view in Pidgin:

```
<Update>
    <data>
        <user>
            <dialogs>/finesse/api/User/1234/Dialogs</dialogs>
            <extension>1001</extension>
            <firstName>John</firstName>
            <lastName>Smith</lastName>
            <loginId>1234</loginId>
            <loginName>jsmith</loginName>
            <roles>
                <role>Agent</role>
            </roles>
            <pendingState></pendingState>
            <reasonCodeId>-1</reasonCodeId>
            <settings>
                <wrapUpOnIncoming></wrapUpOnIncoming>
                <wrapUpOnOutgoing></wrapUpOnOutgoing>
            <settings>
                <state>NOT_READY</state>
                <stateChangeTime>2014-05-27T00:33:44.836Z</stateChangeTime>
                <teamId>1</teamId>
                <teamName>Default</teamName>
                <uri>/finesse/api/User/1234</uri>
            </settings>
        </user>
    </data>
    <event>PUT</event>
    <requestId>xyz</requestId>
    <source>/finesse/api/User/1234</source>
</Update>
```

   The agent is now signed in and in NOT_READY state.

Change Agent State

Use the User - Change agent state API to change the agent state to Ready.

This example uses the same agent information as the previous example.

Note This example shows the URL field for a Unified CCE deployment. In a Unified CCX deployment, you must include the port number in the URL.

1. In Postman, enter the following string in the URL field:

   ```
   https://finesse1.xyz.com/finesse/api/User/1234
   ```

2. Enter the agent's ID (1234) and password (1001) in the two User Auth fields directly under the URL field.
3. In the Content Type field, enter application/XML.
4. In the area under Content Options, enter the following:

   ```
   <User>
    <state>READY</state>
   </User>
   ```

5. (Optional) To add the requestId:
   a. Click **Headers**.
   b. In the Name field, enter **`requestId`**, and in the Value field, enter **`xyz`**.
   c. Click **Add/Change**

6. Click **PUT**.

   Postman returns the following response:

   ```
   PUT on https://finesse1.xyz.com/finesse/api/User/1234
   Status 202: Accepted
   ```

   Finesse returns the following user notification:

   ```
   <Update>
     <data>
       <user>
         <dialogs>/finesse/api/User/1234/Dialogs</dialogs>
         <extension>1001</extension>
         <firstName>John</firstName>
         <lastName>Smith</lastName>
         <loginId>1234</loginId>
         <loginName>jsmith</loginName>
         <roles>
           <role>Agent</role>
         </roles>
         <state>READY</state>
         <pendingState></pendingState>
         <settings>
           <wrapUpOnIncoming></wrapUpOnIncoming>
           <wrapUpOnOutgoing></wrapUpOnOutgoing>
         </settings>
         <stateChangeTime>2014-05-27T00:35:24.123Z</stateChangeTime>
         <teamId>1</teamId>
         <teamName>Default</teamName>
         <uri>/finesse/api/User/1234</uri>
       </user>
     </data>
     <event>PUT</event>
     <requestId>xyz</requestId>
   ```

```
  <source>/finesse/api/User/1234</source>
</Update>
```

CHAPTER 3

Cisco Finesse Desktop APIs

User

The User object represents an agent or supervisor and includes information about the user, such as roles, state, and teams. The User object is structured as follows:

```
<User>
    <uri>/finesse/api/User/1001001</uri>
    <roles>
        <role>Agent</role>
        <role>Supervisor</role>
    </roles>
    <loginId>1001001</loginId>
    <loginName>csmith</loginName>
    <state>NOT_READY</state>
    <stateChangeTime>2012-03-01T17:58:21.234Z</stateChangeTime>
    <mediaType>1</mediaType>
    <pendingState>NOT_READY</pendingState>
    <reasonCodeId>16</reasonCodeId>
    <ReasonCode>
        <category>NOT_READY</category>
        <uri>/finesse/api/ReasonCode/16</uri>
        <code>10</code>
        <label>Team Meeting</label>
        <forAll>true</forAll/>
        <systemCode>false</systemCode>
        <id>16</id>
    </ReasonCode>
    <settings>
```

```
        <wrapUpOnIncoming>OPTIONAL</wrapUpOnIncoming>
        <wrapUpOnOutgoing>NOT_ALLOWED</wrapUpOnOutgoing>
        <deviceSelection>enabled</deviceSelection>
    </settings>
    <extension>1001001</extension>
    <mobileAgent>
        <mode>CALL_BY_CALL</mode>
        <dialNumber>4085551234</dialNumber>
    </mobileAgent>
    <firstName>Chris</firstName>
    <lastName>Smith</lastName>
    <teamId>500</teamId>
    <teamName>Sales</teamName>
    <skillTargetId>6067</skillTargetId>
    <dialogs>/finesse/api/User/1001001/Dialogs</dialogs>
    <teams>
        <Team>
            <uri>/finesse/api/Team/2001</uri>
            <id>2001</id>
            <name>First Line Support</name>
        </Team>
        <Team>
            <uri>/finesse/api/Team/2002</uri>
            <id>2002</id>
            <name>Second Line Support</name>
        </Team>
        <Team>
            <uri>/finesse/api/Team/2003</uri>
            <id>2003</id>
            <name>Third Line Support</name>
        </Team>
      ... other teams ...
    </teams>
    <services>
        <service>AgentAnswers</service>
         ... other services ...
    </services>
    <activeDeviceId>SEP0019305D8EC1</activeDeviceId>
    <Devices>
        <Device>
            <deviceId>SEP0019305D8EC1</deviceId>
            <deviceType>30018</deviceType>
            <deviceTypeName>Cisco 7961</deviceTypeName>
        </Device>
        <Device>
            <deviceId>CSFJP5550016</deviceId>
            <deviceType>503</deviceType>
            <deviceTypeName>Cisco Unified Client Services Framework</deviceTypeName>
        </Device>
    </Devices>
</User>
```

Note The <services> element only applies to Unified CCE deployments.

User APIs

User—Sign In to Finesse

The User—Sign in to Finesse API allows a user to sign in to the CTI server. If the response is successful, the user is signed in to Finesse and is automatically placed in NOT_READY state.

If five consecutive sign-ins fail due to an incorrect password, Finesse blocks access to the user account for a period of 5 minutes.

This API forces a sign-in. That is, if the user is already signed in, that user is authenticated via the sign-in process. If the user's credentials are correct, the user is signed in again but the user keeps the current state. For example, if a user signs in, changes state to Ready, and then signs in again, the user remains in Ready state.

Note To sign in as a mobile agent, see User—Sign In as a Mobile Agent, on page 31.

To sign in to nonvoice Media Routing Domains, see Media—Sign In, on page 191.

URI:	https://<FQDN>/finesse/api/User/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234
Security Constraints:	Users can only act on their own User objects.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<User>`
	`    <state>LOGIN</state>`
	`    <extension>1001001</extension>`
	`    <activeDeviceId>CSFJP5550016</activeDeviceId>`
	`</User>`

Request Parameters:	id (required): The ID of the user state (required): The new state that the user wants to be in (LOGIN) extension (required): The extension with which the user wants to sign in activeDeviceId: A unique ID of the active device. **Note** • This parameter is required when the extension is a shared line between multiple devices and agent is enabled for shared ACD line usage. • This parameter is optional if there is only a single device associated with the extension, while shared ACD line usage is enabled for the agent. • This parameter should not be passed if shared ACD line usage is not enabled for the agent.

HTTP Response:	202: Success 400: Bad Request (for example, malformed or incomplete request, invalid extension) 400: Parameter Missing 401: Unauthorized (for example, the user is not authenticated in the Web Session) 404: Not Found (for example, the user ID is not known) 503: Service Unavailable (for example, the Notification Service is not running)

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>User Not Found</ErrorType>`
	`          <ErrorMessage>UNKNOWN_USER</ErrorMessage>`
	`          <ErrorData>4023</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	User notification

Platform-Based API Differences

Stand-alone Finesse with Unified CCE:

Finesse does not support agent sign-in with an E.164 extension when Finesse is deployed with Unified CCE. However, agents can make calls to and receive calls from E.164 phone numbers.

Coresident Finesse with Unified CCX:

Finesse supports agent sign-in with an E.164 extension when Finesse is deployed with Unified CCX. The maximum number of characters supported for an E.164 extension is 15 (a single plus sign followed by 14 digits).

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Invalid Device	Attempt to sign in an agent with a multiline device without the correct Unified CM configuration for maximum calls and busy trigger for these devices.	All
Invalid Device	Attempt to sign in an agent with a device that does not exist.	All
Invalid Device	Attempt to sign in an agent with a device that is offline.	All
Invalid Device	Attempt to sign in an agent with an extension that is not associated with the Unified CCX Resource Manager provider.	All

ErrorType	Reason	Deployment Type
Device Busy	Attempt to sign in an agent with a device that is already in use.	All
Parameter Missing	Attempt to sign in an agent without an active device, when multiple devices are associated with the extension supplied, while agent is enabled for shared ACD line usage.	All
Invalid Input	Attempt to sign in an agent with an unrecognized or invalid state.	All
Invalid Input	Attempt to sign in an agent with an active device ID,when the protocol version used for CTI connection is lesser than 24. **Note** The protocol version constraint is applicable only for Unified CCE deployments.	All
Authorization Failure	Attempt to sign in an unauthorized agent.	All
Invalid Authorization User Specified	Attempt to make a request by an agent to an unauthorized user.	All
User Not Found	Attempt to sign in an agent with invalid agent ID, or when an agent record is not part of the CTI.	All
Internal Server Error	Attempt to sign in an agent during runtime error (CTI server or any other component).	All

User—Sign In as a Mobile Agent

The User—Sign in as a mobile agent API allows a user to sign in to the CTI server as a mobile agent. This API uses the existing User object with a LOGIN state only. The user must be authenticated to use this API successfully.

If five consecutive sign-ins fail due to an incorrect password, Finesse blocks access to the user account for a period of 5 minutes.

Note Additional configuration is required on Unified CCE and Unified Communications Manager before a mobile agent can sign in. After using this API, you may need to perform additional steps to complete the sign-in. For more information, see the *Cisco Unified Contact Center Enterprise Features Guide*.

Cisco Unified Mobile Agent (Unified MA) enables an agent using an PSTN phone and a broadband VPN connection (for agent desktop communications) to function just like a Unified CCE agent.

URI:	https://<FQDN>/finesse/api/User/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234
Security Constraints:	Users can only act on their own User objects.

HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	`<User> <state>LOGIN</state> <extension>1001001</extension> <mobileAgent> <mode>CALL_BY_CALL</mode> <dialNumber>4085551234</dialNumber> </mobileAgent> </User>`
Request Parameters:	id (required): The ID of the user state (required): The new state that the user wants to be in (for this API, the state must be set to LOGIN) extension (required): The extension with which to sign in the user mobileAgent (required): Indicates that the user is a mobile agent mode (required): The connection mode for the call dialNumber (required): The phone number that the system calls to connect with the mobile agent
HTTP Response:	202: Success This response only indicates the successful completion of the request. The request is processed and the actual response is sent as part of a User notification. 400: Invalid Input (for example, the mode provided is invalid) 400: Parameter Missing (for example the mode or dialNumber was not provided) 400: Generic Error 401: Unauthorized (for example, the user is not authenticated in the Web Session) 401: Invalid User Authorization Specified (an authenticated user tried to make a request for another user) 404: User Not Found (for example, the agent is not recognized)
Example Failure Response:	`<ApiErrors> <ApiError> <ErrorType>Invalid Authorization User Specified</ErrorType> <ErrorData>4321</ErrorData> <ErrorMessage>The user specified in the authentication credentials and the uri don't match</ErrorMessage> </ApiError> </ApiErrors>`
Notifications Triggered:	User notification

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Mode Not Allowed	Attempt to sign in an agent as a mobile agent when that agent is not configured as a mobile agent.	Unified CCE

User—Sign Out of Finesse Desktop

This API allows a user to sign out of Cisco Finesse desktop.

When signing out of the desktop, the user can either sign out of all Media Routing Channels or sign out of configured media channels. Cisco Finesse sends separate sign-out requests to CCE for each MRD.

Note Administrators can use the CLI **utils finesse user_signout_channel** to configure the media channels from which the users are signed out.

For nonvoice MRDs only, users can sign out with active tasks. The user's tasks are either transferred or closed, depending on the way the MRD was configured when the user signed in through the Media - Sign In API.

The desktop sign out fails only if the voice MRD LOGOUT fails; it is not impacted by nonvoice MRD LOGOUT failure.

Note To sign out of nonvoice Media Routing Domains only, see Media—Change State or Sign Out, on page 193.

URI:	https://<FQDN>/finesse/api/User/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234
Security Constraints:	Agents and Supervisors can use this API. Users can only act on their own User objects.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<User>`
	`   <state>LOGOUT</state>`
	`</User>`

Request Parameters:	id (required): The ID of the user state (required): The new state that the user wants to be in (LOGOUT) logoutAllMedia (optional): Determines if the the logout request is for all media channels (true) or only from the channels configured by the Administrator.
HTTP Response:	202: Success 400: Bad Request (for example, malformed or incomplete request, invalid extension) 401: Unauthorized (for example, the user is not authenticated in the Web Session) 404: Not Found (for example, the user ID is not known) 503: Service Unavailable (for example, the Notification Service is not running)

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Invalid Input</ErrorType>`
	`          <ErrorData>state</ErrorData>`
	`         <ErrorMessage>Invalid State specified for user</ErrorMessage>`
	
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	User notification Media notification (for nonvoice MRDs)

Note If a nonvoice MRD signout operation results in an asynchronous error, the error is returned in a Media notification. The notification includes the error type, error code, and error constant. The ErrorMedia parameter indicates the Media RoutingDomain to which the error applies.

User—Get User

The User—Get User API allows a user to get a copy of the User object. For a mobile agent, this operation returns the full User object, including the mobile agent node.

Note Mobile agent information is available to the Cisco Finesse node on which the mobile agent is signed in. However, the other Cisco Finesse node in the cluster does not have the mobile agent information. If the mobile agent signs in to the other node (for example, during a client failover), the mobile agent information is lost and the User object does not return any mobile agent data fields. As a result, the Cisco Finesse desktop inaccurately represents the mobile agent as a regular agent (including all related features). Any other type of CTI failover also results in Cisco Finesse losing the current mobile agent information. However, the Unified Mobile Agent feature behaves as usual whether Cisco Finesse knows that the agent is a mobile agent or not.

As a workaround, the mobile agent can sign out and sign back in as a mobile agent.

URI:	https://<FQDN>/finesse/api/User/<id> For more information on supported characters, see the section "Sign In to Cisco Finesse Desktop" in the *Cisco Finesse Agent and Supervisor Desktop User Guide*.

Example URI:	https://finesse1.xyz.com/finesse/api/User/1234
Security Constraints:	Agents can only get their own User object. Administrators can get any User object. To get the User object, a user must be signed in, or provide valid authorization credentials when challenged.
HTTP Method:	GET
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	—
Request Parameters:	—
HTTP Response:	200: Success 401: Authorization Failure 401: Invalid Authorization User Specified 404: User Not Found 500: Internal Server Error 503: Service Unavailable

Example Response:	

	```
	<User>
	    <uri>/finesse/api/User/1234</uri>
	    <roles>
	        <role>Agent</role>
	        <role>Supervisor</role>
	    </roles>
	    <loginId>1234</loginId>
	    <loginName>csmith</loginName>
	    <state>NOT_READY</state>
	    <stateChangeTime>2012-03-01T17:58:21.234Z</stateChangeTime>
	    <pendingState></pendingState>
	    <reasonCodeId>16</reasonCodeId>
	    <ReasonCode>
	        <category>NOT_READY</category>
	        <uri>/finesse/api/ReasonCode/16</uri>
	        <code>10</code>
	        <label>Team Meeting</label>
	        <forAll>true</forAll>
	        <id16</id>
	    </ReasonCode>
	    <settings>
	        <wrapUpOnIncoming>OPTIONAL</wrapUpOnIncoming>
	        <wrapUpOnOutgoing>REQUIRED</wrapUpOnOutgoing>
	        <deviceSelection>enabled</deviceSelection>
	    </settings>
	    <extension>1001001</extension>
	    <mobileAgent>
	        <mode>CALL_BY_CALL</mode>
	        <dialNumber>4085551234</dialNumber>
	    </mobileAgent>
	    <firstName>Chris</firstName>
	    <lastName>Smith</lastName>
	    <teamId>500</teamId>
	    <teamName>Sales</teamName>
	    <skillTargetId>6067</skillTargetId>
	    <dialogs>/finesse/api/User/1234/Dialogs</dialogs>
	    <teams>
	        <Team>
	            <uri>/finesse/api/Team/2001</uri>
	            <id>2001</id>
	            <name>First Line Support</name>
	        </Team>
	        <Team>
	            <uri>/finesse/api/Team/2002</uri>
	            <id>2002</id>
	            <name>Second Line Support</name>
	        </Team>
	        <Team>
	            <uri>/finesse/api/Team/2003</uri>
	            <id>2003</id>
	            <name>Third Line Support</name>
	        </Team>
	... other teams ...
	    </teams>
	    <activeDeviceId>SEP0019305D8EC1</activeDeviceId>
	    <Devices>
	        <Device>
	            <deviceId>SEP0019305D8EC1</deviceId>
	            <deviceType>30018</deviceType>
	            <deviceTypeName>Cisco 7961</deviceTypeName>
	        </Device>
	        <Device>
	            <deviceId>CSFJP5550016</deviceId>
	            <deviceType>503</deviceType>
	```

	`            <deviceTypeName>Cisco Unified Client Services Framework</deviceTypeName>`
	`        </Device>`
	`    </Devices>`
	`</User>`

Example Response (Mobile Agent):
Note Mobile agent only applies to Unified CCE deployments).
	`<User>`
	`   ... Full User Object ...`
	`   <mobileAgent>`
	`      <mode>CALL_BY_CALL</mode>`
	`      <dialNumber>4085551234</dialNumber>`
	`   </mobileAgent>`
	`</User>`

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>User Not Found</ErrorType>`
	`          <ErrorMessage>UNKNOWN_USER</ErrorMessage>`
	`          <ErrorData>4023</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

User—Get User Id from loginName

The User—Get User Id from loginName API accepts the loginName in the URI and authentication for both SSO and non-SSO deployments. This API is only supported for Unified CCE deployments.

In Unified CCE, an agent is assigned with an AgentID (peripheral number) and a Login name, but they are different from one another.

Use the User—Get User Id from loginName API to retrieve the agent's peripheral ID from the LoginName.

Clients in Unified CCE SSO deployments can use the User—Get API request to retrieve the peripheralID using the username obtained from the Cisco Identity Service (IdS) token. The userName has to be URL encoded with UTF-8.

URI:	**For Unified CCE**: https://<FQDN>/finesse/api/User/<loginName> For more information on supported characters, see the section "Sign In to Cisco Finesse Desktop" in the *Cisco Finesse Agent and Supervisor Desktop User Guide*.
Example URI:	https://finesse1.xyz.com/finesse/api/User/csmith
Security Constraints:	Agents can only get their own User object. Administrators can get any User object. To get the User object, a user must be signed in, or provide valid authorization credentials when challenged.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
Request Parameters:	—

HTTP Response:	200: Success 401: Authorization Failure 401: Invalid Authorization User Specified 404: User Not Found 500: Internal Server Error 503: Service Unavailable

Example Response:

```
<User>
    <uri>/finesse/api/User/1234</uri>
    <roles>
        <role>Agent</role>
        <role>Supervisor</role>
    </roles>
    <loginId>1234</loginId>
    <loginName>csmith</loginName>
    <state>NOT_READY</state>
    <stateChangeTime>2012-03-01T17:58:21.234Z</stateChangeTime>
    <pendingState></pendingState>
    <reasonCodeId>16</reasonCodeId>
    <ReasonCode>
        <category>NOT_READY</category>
        <uri>/finesse/api/ReasonCode/16</uri>
        <code>10</code>
        <label>Team Meeting</label>
        <forAll>true</forAll>
        <id16</id>
    </ReasonCode>
    <settings>
        <wrapUpOnIncoming>OPTIONAL</wrapUpOnIncoming>
        <wrapUpOnOutgoing>REQUIRED</wrapUpOnOutgoing>
    </settings>
    <extension>1001001</extension>
    <mobileAgent>
        <mode>CALL_BY_CALL</mode>
        <dialNumber>4085551234</dialNumber>
    </mobileAgent>
    <firstName>Chris</firstName>
    <lastName>Smith</lastName>
    <teamId>500</teamId>
    <teamName>Sales</teamName>
    <dialogs>/finesse/api/User/1234/Dialogs</dialogs>
    <teams>
        <Team>
            <uri>/finesse/api/Team/2001</uri>
            <id>2001</id>
            <name>First Line Support</name>
        </Team>
        <Team>
            <uri>/finesse/api/Team/2002</uri>
            <id>2002</id>
            <name>Second Line Support</name>
        </Team>
        <Team>
            <uri>/finesse/api/Team/2003</uri>
            <id>2003</id>
            <name>Third Line Support</name>
        </Team>
... other teams ...
    </teams>
</User>
```

Example Failure Response:

```
<ApiErrors>
     <ApiError>
          <ErrorType>User Not Found</ErrorType>
          <ErrorMessage>UNKNOWN_USER</ErrorMessage>
          <ErrorData>4023</ErrorData>
   </ApiError>
</ApiErrors>
```

User—Get List

This API allows an administrator to get a list of users.

URI:	https://<FQDN>/finesse/api/Users
Example URI:	https://finesse1.xyz.com/finesse/api/Users
Security Constraints:	Only administrators can get a list of users. To get a list of users, the administrator must be signed in or provide valid authorization credentials when challenged. If this API is accessed through a reverse-proxy, the response is provided without any authentication.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 500: Internal Server Error 503: Service Unavailable

Example Response:	`<Users>`
	`   <User>`
	`      ... Full User Object ...`
	`   </User>`
	`   <User>`
	`      ... Full User Object ...`
	`   </User>`
	`   <User>`
	`      ... Full User Object ...`
	`   </User>`
	`   <User>`
	`      ... Full User Object ...`
	`   </User>`
	`   <User>`
	`      ... Full User Object ...`
	`   </User>`
	`      ... Additional Users...`
	`</Users>`

Example Failure Response:	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Unauthorized</ErrorType>`
	`    <ErrorMessage>The user is not authorized to perform`
	`     this operation</ErrorMessage>`
	`  </ApiError>`
	`</ApiErrors>`

User—Get List of Dialogs (Voice Only by Default)

This API allows an agent or administrator to get a list of dialogs associated with a particular user. By default, this API returns voice dialogs only. You can use the query parameters to include nonvoice dialogs.

The URI for this API contains two query parameters:

- **type:** (optional) Set the type to return voice or nonvoice dialogs for a user. You can include both types to return all dialogs for a user (type=voice&type=non-voice). If you do not include the type query parameter, only voice dialogs are returned.
- **media:** (optional) Use this parameter to filter nonvoice dialog results by a specific media id. This parameter is only applicable when the "type=non-voice" query parameter is used.

URI:	https://<FQDN>/finesse/api/User/<id>/Dialogs?type={voice|non-voice}&media={id}
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Dialogs
Security Constraints:	Agents can only get a list of their own dialogs, supervisors can get a list of dialogs associated to the agents in their teams, and administrators can get a list of dialogs associated with any user. To get a list of dialogs, a user must be signed in or provide valid authorization credentials when challenged.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 500: Internal Server Error 503: Service Unavailable

Example Response:

```
<Dialogs>
    <Dialog>
       ... Full Dialog Object ...
    </Dialog>
    <Dialog>
       ... Full Dialog Object ...
    </Dialog>
    <Dialog>
       ... Full Dialog Object ...
    </Dialog>
    <Dialog>
       ... Full Dialog Object ...
    </Dialog>
    <Dialog>
       ... Full Dialog Object ...
    </Dialog>
       ... Additional Dialogs...
</Dialogs>
```

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

User—Get List of Dialogs (Nonvoice Only)

This API allows an agent or administrator to get a list of nonvoice dialogs associated with a particular user for a specific Media Routing Domain (MRD).

URI:	https://<FQDN>/finesse/api/User/<id>/Media/<mrdId>/Dialogs
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Media/5001/Dialogs
Security Constraints:	Agents can only get a list of their own dialogs. Administrators can get a list of dialogs associated with any user. To get a list of dialogs, a user must be signed in or provide valid authorization credentials when challenged.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 500: Internal Server Error 503: Service Unavailable

Example Response:	`<Dialogs>`
	`    <Dialog>`
	`       ... Full Dialog Object ...`
	`    </Dialog>`
	`    <Dialog>`
	`       ... Full Dialog Object ...`
	`    </Dialog>`
	`    <Dialog>`
	`       ... Full Dialog Object ...`
	`    </Dialog>`
	`    <Dialog>`
	`       ... Full Dialog Object ...`
	`    </Dialog>`
	`    <Dialog>`
	`       ... Full Dialog Object ...`
	`    </Dialog>`
	`       ... Additional Dialogs...`
	`</Dialogs>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

User—Get List of Reservation Dialogs

This API allows an agent or administrator to get a list of reservation dialogs and is applicable for progressive and predictive outbound reservation calls.

URI:	https://<FQDN>/finesse/api/User/<id>/ReservationDialogs
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/ReservationDialogs
Security Constraints:	Agents can get a list of their outbound reservation dialogs. Administrators can get a list of outbound reservation dialogs for all the users. To get a list of outbound reservation dialogs, a user must be signed in or must have the valid authorization credentials.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Invalid Authorization 500: Internal Server Error 503: Service Unavailable

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

User—Change Agent State

This API allows a user to change the state of an agent on the CTI server. Agents can change their own states

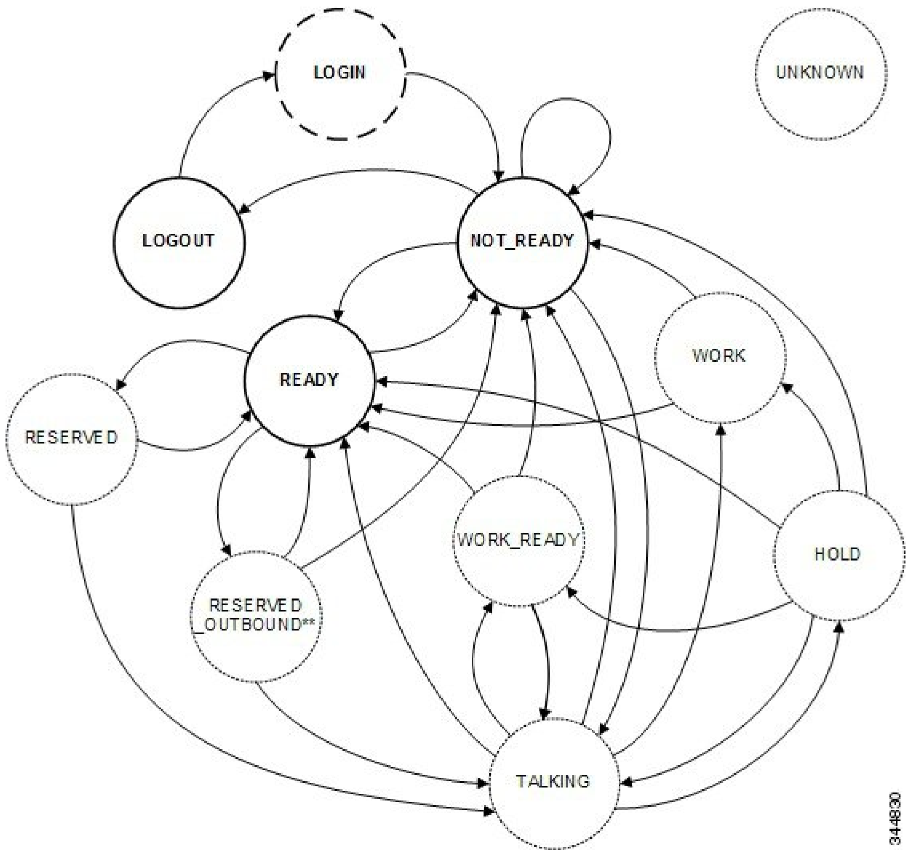
Note To change user state in a nonvoice Media Routing Domain, see Media—Change State or Sign Out, on page 193.

If the request to change an agent's state is successful, the response is sent as part of a User notification.

The following figure illustrates the supported state transitions by Unified CCE agents.

Note The following diagram contains only logical state transitions. Because the underlying system determines the state, an agent can transition from any state to any state, especially under failover conditions. The diagram describes the typical state changes that occur in the system.

Figure 10: Supported State Transitions by Agent (Unified CCE)



Note In the preceding diagram, RESERVED_OUTBOUND can represent RESERVED_OUTBOUND or RESERVED_OUTBOUND_PREVIEW state.

The following table describes supported agent state transitions for Unified CCE.

From	To	Description
*	UNKNOWN	If the agent state is unknown, the state is UNKNOWN. This scenario is unlikely.

From	To	Description
LOGOUT	LOGIN	To sign in to Finesse, the agent sets the state to LOGIN. LOGIN is a transient state and transitions to NOT_READY.
LOGIN	NOT_READY	After a successful LOGIN, the agent transitions to NOT_READY.
NOT_READY	LOGOUT	To sign out of Finesse, the agent sets the state to LOGOUT. An agent can set the state to LOGOUT only if that agent is in NOT_READY state.
NOT_READY	NOT_READY	To change their Not Ready reason code, agents can set a NOT_READY state from NOT_READY.
NOT_READY	READY	To become available for incoming or Outbound Option calls, agents set their state to READY.
NOT_READY	TALKING	An agent who places a call while in NOT_READY state transitions to TALKING.
READY	RESERVED	An incoming call arrives at an agent.
READY	RESERVED _OUTBOUND	An outbound agent becomes reserved to handle an Outbound Option Progressive or Predictive call.
READY	RESERVED_OUTBOUND _PREVIEW	An outbound agent becomes reserved to handle an Outbound Option Preview call.
READY	NOT_READY	Agents can change to NOT_READY to make themselves unavailable for incoming calls.
RESERVED	READY	An agent can become RESERVED but never take a call.
RESERVED	TALKING	When an agent answers an incoming call, the agent transitions to TALKING.
RESERVED _OUTBOUND	READY	An agent can change to READY state to leave RESERVED_OUTBOUND. If the system deems it necessary, that agent may transition back to RESERVED_OUTBOUND.
RESERVED _OUTBOUND	NOT_READY	An agent can change to NOT_READY state to leave RESERVED_OUTBOUND.
RESERVED _OUTBOUND	TALKING	An agent transitions to TALKING when an Outbound Option call arrives at the agent.
RESERVED_OUTBOUND _PREVIEW	READY	An agent transitions to READY if the agent was in READY state before being reserved in an Outbound Option Preview campaign.

From	To	Description
RESERVED_OUTBOUND _PREVIEW	NOT_READY	An agent transitions to NOT_READY if that agent changes state to NOT_READY while reserved in an Outbound Option Preview campaign. This state change is a pending state change. The agent does not transition to NOT_READY until the call is complete or the Outbound Option Preview reservation is closed or rejected.
RESERVED_OUTBOUND _PREVIEW	TALKING	An agent transitions to TALKING when an Outbound Option call arrives at the agent.
TALKING	READY	If an agent is on a call that is dropped, the agent transitions to READY (if the agent was in READY state before the call).
TALKING	NOT_READY	If an agent is on a call that is dropped, the agent transitions to NOT_READY if that agent was in NOT_READY state before the call.
TALKING	WORK	If wrap-up is enabled, and the agent chooses NOT_READY while on a call, that agent enters WORK state after the call is dropped.
TALKING	WORK_READY	If wrap-up is enabled, an agent enters WORK_READY state after a call is dropped.
TALKING	HOLD	An agent puts a call on hold and transitions to HOLD state.
HOLD	READY	If an agent is connected to a held call and the call is dropped, the agent transitions to READY state (if the agent was in READY state before the call).
HOLD	NOT_READY	If an agent is connected to a held call and the call is dropped, the agent transitions to NOT_READY state (if the agent was in NOT_READY state before the call).
HOLD	WORK	If wrap-up is enabled and an agent is connected to a held call that is dropped, the agent transitions to WORK state if the agent chose to go NOT_READY during the call.
HOLD	WORK_READY	If wrap-up is enabled and an agent is connected to a held call that is dropped, the agent transitions to WORK_READY state.
HOLD	TALKING	When an agent retrieves a held call, the agent transitions to TALKING state.
WORK	READY	To leave WORK state, agents can set their state to READY.

From	To	Description
WORK	NOT_READY	To leave WORK state, agents can set their state to NOT_READY. Agents automatically transition to NOT_READY after the wrap-up timer expires.
WORK_READY	READY	To leave WORK_READY state, agents can set their state to READY. Agents automatically transition to READY after the wrap-up timer expires.
WORK_READY	NOT_READY	To leave WORK_READY state, agents can set their state to NOT_READY.

The following table describes supported agent state transitions for Unified CCX.

From	To	Description
LOGIN	NOT_READY	After a successful LOGIN, the agent transitions to NOT_READY.
NOT_READY	LOGOUT	To sign out of Finesse, the agent sets the state to LOGOUT.
NOT_READY	NOT_READY	To change their Not Ready reason code, agents can set a NOT_READY state from NOT_READY.
NOT_READY	READY	To become available for incoming calls, agents set their state to READY.
READY	NOT_READY	Agents can change their state to NOT_READY to make themselves unavailable for incoming calls.
READY	LOGOUT	To sign out of Finesse, agents set their state to LOGOUT.
READY	RESERVED_ OUTBOUND_ PREVIEW	An outbound agent becomes reserved to handle an Outbound Option Direct Preview call.
RESERVED_ OUTBOUND_ PREVIEW	TALKING	An outbound agent accepts a direct preview call and the call is active.

Users can set the following states with this API:

- READY
- NOT_READY
- LOGOUT

The LOGIN state is a transitive state. That is, when set, LOGIN triggers a change that results in a new state.

Users can be in the following states while on a call. However, users cannot place themselves in these states. For example, agents cannot change their state to TALKING. Agents enter TALKING state when they answer a call.

- RESERVED
- RESERVED_OUTBOUND
- RESERVED_OUTBOUND_PREVIEW
- TALKING
- HOLD
- WORK
- WORK_READY

RESERVED_OUTBOUND user state:

Users who belong to Outbound Option skill groups transition from READY state to RESERVED_OUTBOUND state when those users are reserved for Progressive or Predictive Outbound Option calls.

In a Unified CCE deployment, users can change their state to READY or NOT_READY to exit this state. If not ready reason codes are configured, users must specify a reason code to transition to NOT_READY state. If the user does nothing and then the call is transferred to the user, the user transitions to TALKING state. If the call is not transferred to the user, the user transitions back to READY state.

In a Unified CCX deployment, users cannot change their state to exit RESERVED_OUTBOUND state. If auto-answer for the predictive or progressive call is not enabled and the agent does not answer the call, the agent transitions to NOT_READY state. If the call does not reach a voice contact or if the reservation timer on Unified CCX expires, the agent transitions to READY state.

RESERVED_OUTBOUND_PREVIEW user state:

Users who belong to Outbound Option skill groups transition from READY state to RESERVED_OUTBOUND_PREVIEW state when they are reserved for Outbound Option Preview or Direct Preview calls. Users cannot set their state to RESERVED_OUTBOUND_PREVIEW.

In a Unified CCE deployment, users can click Close or Reject on the Outbound Option dialog. Changing the user's state to READY or NOT_READY does not generate a state change notification but does affect the user state when the call is complete. For example, if the user selects NOT_READY state while in RESERVED_OUTBOUND_PREVIEW state, the user transitions to NOT_READY state after clicking Close or Reject.

In a Unified CCX deployment, users cannot change their state directly when in RESERVED_OUTBOUND_PREVIEW state. The state can only be changed by issuing a Dialog Accept, Close, or Reject request or when the reservation call times out.

WORK and WORK_READY user states:

A user is in WORK or WORK_READY state during wrap-up. A user is placed in WORK state when the user is set to transition to NOT_READY state when wrap-up ends. A user is in WORK_READY state when the user is set to transition to READY state when wrap-up ends.

A user transitions to WORK state for the following reasons:

- The user was in NOT_READY state before taking a call.
- The user set a state of NOT_READY while in TALKING state.

When the wrap-up timer expires, the user transitions to NOT_READY state.

WORK_READY state applies only to Unified CCE deployments. A user transitions to WORK_READY state for the following reasons:

- The user was in READY state before taking a call.
- The user set a state of READY while in TALKING state.

When the wrap-up timer expires, the user transitions to READY state.

Note The following statements apply to a supervisor using this API to change the state of an agent or other supervisor:

- A supervisor can only change the state of a user who is assigned to that supervisor's team.
- A supervisor can only set the state of another user to NOT_READY, READY, or LOGOUT.
- A supervisor can set the state of a user to LOGOUT only if that user is in READY, NOT_READY, RESERVED, RESERVED_OUTBOUND, RESERVED_OUTBOUND_PREVIEW, TALKING, HOLD, WORK, or WORK_READY state.
- A supervisor can set the state of a user to NOT_READY only if that user is in READY, WORK, or WORK_READY state.
- When a supervisor uses this API to set the state of a user to NOT_READY, a reason code must not be used. If a reason code is provided, Finesse rejects it and returns a 400 Invalid Input error. Finesse sends a hard-coded reason code to indicate that the state change was performed by the supervisor.

URI:	https://<FQDN>/finesse/api/User/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234
Security Constraints:	Agents can only act on their own User objects. Supervisors can act on the User objects of agents who belong to their team.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<User>`
	`   <state>READY</state>`
	`</User>`

Request Parameters:	id (required): The ID of the user state (required): The new state the user wants to be in (for example, LOGOUT, READY, NOT_READY)

HTTP Response:	200: Success 400: Bad Request 401: Invalid Supervisor 401: Unauthorized 404: Not Found 500: Internal Server Error 503: Service Unavailable

Example Failure Response:	`<ApiErrors>`
	  `<ApiError>`
	    `<ErrorType>Parameter Missing</ErrorType>`
	    `<ErrorData>state</ErrorData>`
	    `<ErrorMessage>State Parameter missing</ErrorMessage>`
	  `</ApiError>`
	`</ApiErrors>`

Notifications Triggered:	User notification

Platform-Based API Differences

The following table describes API differences between a stand-alone Finesse deployment with Unified CCE and a coresident Finesse deployment with Unified CCX.

Scenario	Response

Change from LOGOUT to NOT_READY.	**Stand-alone Finesse with Unified CCE:**
	`<data>`
	    `<apiErrors>`
	        `<apiError>`
	            `<errorData>257</errorData>`
	          `<errorMessage>CF_INVALID_PASSWORD_SPECIFIED</errorMessage>`
	
	            `<errorType>Invalid State</errorType>`
	        `</apiError>`
	    `</apiErrors>`
	`</data>`
	Coresident Finesse with Unified CCX:
	`<data>`
	    `<apiErrors>`
	        `<apiError>`
	            `<errorData>1010</errorData>`
	            `<errorMessage>CF_INVALID_PARAMETER</errorMessage>`
	            `<errorType>Invalid State</errorType>`
	        `</apiError>`
	    `</apiErrors>`
	`</data>`

Scenario	Response
Agent receives and answers a non-ICD call.	**Stand-alone Finesse with Unified CCE:** Finesse sends a User notification with state=TALKING. **Coresident Finesse with Unified CCX:** Finesse does not send a User notification. The agent remains in NOT_READY state.
Agent puts an ICD call on hold.	**Stand-alone Finesse with Unified CCE:** Finesse sends a User notification with state=HOLD. **Coresident Finesse with Unified CCX:** Finesse does not send a User notification. The agent remains in TALKING state.

While talking on an ICD call, the agent sets a pending state of READY.	**Stand-alone Finesse with Unified CCE:**
	Agent transitions to READY state after the call ends.
	Coresident Finesse with Unified CCX:
	Unified CCX does not allow an agent to set a pending state of READY while that agent is talking on an ICD call.
	`<data>`
	`    <apiErrors>`
	`        <apiError>`
	`            <errorData>265</errorData>`
	`            <errorMessage>CF_INVALID_AGENT_WORKMODE</errorMessage>`
	`            <errorType>Invalid State</errorType>`
	`        </apiError>`
	`    </apiErrors>`
	`</data>`

While talking on a non-ICD call (agent state can be TALKING in Unified CCE or NOT_READY in Unified CCX), the agent sets a pending state of READY.	**Stand-alone Finesse with Unified CCE:**
	Agent transitions to READY state after the call ends.
	Coresident Finesse with Unified CCX:
	Unified CCX does not allow an agent to set a pending state of READY while that agent is talking on a non-ICD call.
	`<data>`
	`    <apiErrors>`
	`        <apiError>`
	`            <errorData>33</errorData>`
	`            <errorMessage>CF_RESOURCE_BUSY</errorMessage>`
	`            <errorType>Invalid State</errorType>`
	`        </apiError>`
	`    </apiErrors>`
	`</data>`

Scenario	Response

While talking on an ICD call, the agent attempts to change from a pending state of NOT_READY with reason code 1 to a pending state of NOT_READY with reason code 2.	**Stand-alone Finesse with Unified CCE:**
	Agent transitions to NOT_READY state with reason code 2 after the call ends.
	Coresident Finesse with Unified CCX:
	Unified CCX allows an agent to set a pending state of NOT_READY only once during a call. Unified CCX does not allow an agent to change from one Not Ready reason code to another.
	`<data>`
	`    <apiErrors>`
	`        <apiError>`
	`            <errorData>265</errorData>`
	`            <errorMessage>CF_INVALID_AGENT_WORKMODE</errorMessage>`
	`            <errorType>Invalid State</errorType>`
	`        </apiError>`
	`    </apiErrors>`
	`</data>`

A supervisor changes the state of an agent on that supervisor's team to NOT_READY.	**Stand-alone Finesse with Unified CCE:** Finesse sends a hard-coded reason code of 999 to indicate the forced state change. **Coresident Finesse with Unified CCX:** Finesse sends a hard-coded reason code of 33 to indicate the forced state change.

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Invalid State	Invalid state transition requested. For example, attempt to set Wrap-Up state on an agent that is not allowed to go to Wrap-Up, or attempt to change an agent's state from READY state to Wrap-up or WORK state.	All
Internal Server Error	Attempt to change an agent's state from RESERVED_OUTBOUND to any other state.	Unified CCX

User—Agent State Change With Reason Code

This API allows a user to change the agent state in the CTI server and pass along the code value of a corresponding reason code. Users can use this API only when changing state to NOT_READY or LOGOUT.

If the user is changing state to LOGOUT and is signing out of all Media Routing Domains, the same reason code is applied to all the Media Routing Domains.

Note To change state with a reason code in a nonvoice Media Routing Domain only, see Media—Change Agent State with Reason Code, on page 194.

URI:	https://<FQDN>/finesse/api/User/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234
Security Constraints:	Users can only act on their own User objects.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<User>`
	`   <state>LOGOUT</state>`
	`   <reasonCodeId>10</reasonCodeId>`
	`</User>`

Request Parameters:	id (required): The ID of the user reasonCodeID (required if reason codes are configured for the given state): The database ID for the reason code state (required): The new state the user wants to be in (NOT_READY, LOGOUT) logoutAllMedia (optional): This parameter can be included if changing the state to LOGOUT. When the user signs out of Cisco Finesse desktop, the parameter LogoutAllMedia determines whether the user signs out from all Media Routing Domains or only from the configured domains. If the parameter LogoutallMedia is set to true, then users are signed from all the media channels. If set to false or the value is not specified, then based on the values configured by the Administrator for the CLI **utils finesse user_signout_channel** users are signed out from respective channels.
HTTP Response:	202: Successfully Accepted 400: Parameter Missing 400: Invalid Input 400: Invalid State 401: Authorization Failure (for example, the user is not authenticated in the Web Session) 401: Invalid Authorization Specified (for example, the authenticated user tried to make a request for another user)

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Parameter Missing</ErrorType>`
	`      <ErrorData>state</ErrorData>`
	`      <ErrorMessage>State Parameter missing</ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	User notification Media notification (for nonvoice MRDS, when changing state to LOGOUT)

Note If a nonvoice MRD sign out operation results in an asynchronous error, the error is returned in a Media notification. The notification includes the error type, error code, and error constant. The ErrorMedia parameter indicates the Media RoutingDomain to which the error applies.

User—Get Reason Code

This API allows an agent or supervisor to get an individual Not Ready or Sign Out reason code, which is already defined and stored in the Finesse database (and that is applicable to the agent or supervisor).

Users can select the reason code to display on their desktops when they change their state to NOT_READY or LOGOUT.

For more information about the ReasonCode object, see section on *ReasonCode*.

URI:	https://<FQDN>/finesse/api/User/<id>/ReasonCode/<reasonCodeId>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/ReasonCode/12
Security Constraints:	Administrators, agents, and supervisors can use this API. To get a reason code, a user must be signed in or provide valid authorization credentials when challenged. The reason code must be global (forAll parameter set to true) or be assigned to a team to which the user belongs. Only an administrator can get another user's reason codes.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error (for example, the object does not exist, the object is stale, or violation of DB constraint) 401: Authorization Failure 401: Invalid Authorization User Specified 404: Not Found (for example, the reason code does not exist or has been deleted) 500: Internal Server Error

Example Response:	`<ReasonCode>`
	`   <uri>finesse/api/ReasonCode/1</uri>`
	`   <category>NOT_READY</category>`
	`   <code>12</code>`
	`   <label>Lunch</label>`
	`   <forAll>true</forAll>`
	`</ReasonCode>`

Example Failure Response:	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Authorization Failure</ErrorType>`
	`    <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`    <ErrorData>1234</ErrorData>`
	`  </ApiError>`
	`</ApiErrors>`

User—Get Reason Code List

This API allows an agent or supervisor to get a list of Not Ready or Sign Out reason codes (that are applicable to that agent or supervisor), which are defined and stored in the Finesse database. Users can assign one of the reason codes on the desktop when they change their state to NOT_READY or LOGOUT. Cisco Finesse Release 12.5(1) onward, this API is deprecated.

Note The ReasonCode list can be empty (for example, if no reason codes for the specified category exist in the Finesse configuration database).

Reason codes that have the forAll parameter set to true apply to any user.

The category parameter is required when making a request to get a list of reason codes.

For information about the ReasonCode object, see section on *ReasonCode*.

URI:	https://<FQDN>/finesse/api/User/<id>/ReasonCodes?category=NOT_READY|LOGOUT
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/ReasonCodes?category=NOT_READY
Security Constraints:	Administrators, agents and supervisors can use this API. To get a list of reason codes, a user must be signed in or provide valid authorization credentials when challenged. Only an administrator can get another user's list of reason codes.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error (for example, the object does not exist, the object is stale, or violation of DB constraint) 401: Authorization Failure 401: Invalid Authorization User Specified 404: Not Found 500: Internal Server Error

Example Response:	`<ReasonCodes category="NOT_READY">`
	`    <ReasonCode>`
	`        <uri>/finesse/api/ReasonCode/1</uri>`
	`        <category>NOT_READY</category>`
	`        <code>12</code>`
	`        <label>Lunch</label>`
	`        <forAll>true</forAll>`
	`    </ReasonCode>`
	`    <ReasonCode>`
	`      ...Full ReasonCode Object...`
	`   </ReasonCode>`
	`    <ReasonCode>`
	`      ...Full ReasonCode Object...`
	`   </ReasonCode>`
	`</ReasonCodes>`

Example Failure Response:	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Authorization Failure</ErrorType>`
	`    <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`    <ErrorData>1234</ErrorData>`
	`  </ApiError>`
	`</ApiErrors>`

User—Get Wrap-Up Reason

This API allows a user to get a WrapUpReason object.

For more information about the WrapUpReason object, see WrapUpReason, on page 267.

URI:	https://<FQDN>/finesse/api/User/<id>/WrapUpReason/<wrapUpReasonId>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/WrapUpReason/1001
Security Constraints:	Administrators, agents, and supervisors can use this API. To get a wrap-up reason, a user must be signed in, or provide valid authorization credentials when challenged. Only an administrator can get another user's wrap-up reasons.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML

HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request (the request body is invalid) 400: Finesse API Error (for example, the object does not exist, the object is stale, or violation of DB constraint) 401: Authorization Failure 401: Invalid Authorization User Specified 404: Not Found (for example, the wrap-up reason does not exist or has been deleted) 500: Internal Server Error

Example Response:	`<WrapUpReason>`
	`   <uri>finesse/api/User/1234/WrapUpReason/205</uri>`
	`   <label>Product Question</label>`
	`   <forAll>true</forAll>`
	`</WrapUpReason>`

Example Failure Response:	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Authorization Failure</ErrorType>`
	`    <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`    <ErrorData>1234</ErrorData>`
	`  </ApiError>`
	`</ApiErrors>`

User—Get Wrap-Up Reason List

This API allows a user to get a list of all wrap-up reasons applicable for that user. Cisco Finesse Release 12.5(1) onward, this API is deprecated.

For more information about the WrapUpReason object, see WrapUpReason, on page 267.

URI:	https://<FQDN>/finesse/api/User/<id>/WrapUpReasons
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/WrapUpReasons
Security Constraints:	Administrators, agents, and supervisors can use this API. To get a list of wrap-up reasons, a user must be signed in or provide valid authorization credentials when challenged. Only an administrator can get another user's list of wrap-up reasons.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Finesse API Error (for example, the object does not exist, the object is stale, or violation of DB constraint) 401: Authorization Failure 401: Invalid Authorization User Specified 404: User Not Found 500: Internal Server Error

Example Response:	`<WrapUpReasons>`
	`  <WrapUpReason>`
	`    <label>Successful tech support call</label>`
	`    <forAll>true</forAll>`
	`    <uri>/finesse/api/User/1234/WrapUpReason/12</uri>`
	`  </WrapUpReason>`
	`    ... more wrap-up reasons ...`
	`</WrapUpReasons>`

Example Failure Response:	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Authorization Failure</ErrorType>`
	`    <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`    <ErrorData>1234</ErrorData>`
	`  </ApiError>`
	`</ApiErrors>`

User—Get Default Media Properties Layout

This API allows a user to get a copy of the default MediaPropertiesLayout object. The MediaPropertiesLayout object determines how call variables and ECC variables appear on the Finesse desktop.

URI:	https://<FQDN>/finesse/api/User/<id>/MediaPropertiesLayout
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/MediaPropertiesLayout
Security Constraints:	Agents and supervisors can use this API. To get the default MediaPropertiesLayout object, a user must be signed in or provide valid authorization credentials when challenged.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 500: Internal Server Error

Example Response:	

	```
	<MediaPropertiesLayout>
	  <header>
	    <entry>
	      <displayName>Call Variable 1</displayName>
	      <mediaProperty>callVariable1</mediaProperty>
	    </entry>
	  </header>
	  <column>
	    <entry>
	      <displayName>BA AccountNumber</displayName>
	      <mediaProperty>BAAccountNumber</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>BA Campaign</displayName>
	      <mediaProperty>BACampaign</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 1</displayName>
	      <mediaProperty>callVariable1</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 2</displayName>
	      <mediaProperty>callVariable2</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 3</displayName>
	      <mediaProperty>callVariable3</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 4</displayName>
	      <mediaProperty>callVariable4</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 5</displayName>
	      <mediaProperty>callVariable5</mediaProperty>
	    </entry>
	  </column>
	  <column>
	    <entry>
	      <displayName>BA Status</displayName>
	      <mediaProperty>BAStatus</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>BA Response</displayName>
	      <mediaProperty>BAResponse</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 6</displayName>
	      <mediaProperty>callVariable6</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 7</displayName>
	      <mediaProperty>callVariable7</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 8</displayName>
	      <mediaProperty>callVariable8</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 9</displayName>
	      <mediaProperty>callVariable9</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 10</displayName>
	```

	```
	      <mediaProperty>callVariable10</mediaProperty>
	    </entry>
	  </column>
	  <uri>/finesse/api/MediaPropertiesLayout/1</uri>
	  <name>Default Layout</name>
	  <description>Layout used when no other layout matches the user layout
	Custom/ECC Variable</description>
	  <type>DEFAULT</type>
	</MediaPropertiesLayout>
	<MediaPropertiesLayout>
	  <header>
	    <entry>
	      <displayName>Call Variable 1</displayName>
	      <mediaProperty>callVariable1</mediaProperty>
	    </entry>
	  </header>
	  <column>
	    <entry>
	      <displayName>BA AccountNumber</displayName>
	      <mediaProperty>BAAccountNumber</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>BA Campaign</displayName>
	      <mediaProperty>BACampaign</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 1</displayName>
	      <mediaProperty>callVariable1</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 2</displayName>
	      <mediaProperty>callVariable2</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 3</displayName>
	      <mediaProperty>callVariable3</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 4</displayName>
	      <mediaProperty>callVariable4</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 5</displayName>
	      <mediaProperty>callVariable5</mediaProperty>
	    </entry>
	  </column>
	  <column>
	    <entry>
	      <displayName>BA Status</displayName>
	      <mediaProperty>BAStatus</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>BA Response</displayName>
	      <mediaProperty>BAResponse</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 6</displayName>
	      <mediaProperty>callVariable6</mediaProperty>
	    </entry>
	    <entry>
	      <displayName>Call Variable 7</displayName>
	      <mediaProperty>callVariable7</mediaProperty>
	    </entry>
	    <entry>
	```

	`<displayName>Call Variable 8</displayName>`
	`<mediaProperty>callVariable8</mediaProperty>`
	`</entry>`
	`<entry>`
	`<displayName>Call Variable 9</displayName>`
	`<mediaProperty>callVariable9</mediaProperty>`
	`</entry>`
	`<entry>`
	`<displayName>Call Variable 10</displayName>`
	`<mediaProperty>callVariable10</mediaProperty>`
	`</entry>`
	`</column>`
	`<uri>/finesse/api/MediaPropertiesLayout/1</uri>`
	`<name>Default Layout</name>`
	`<description>Layout used when no other layout matches the user layout Custom/ECC Variable</description>`
	`<type>DEFAULT</type>`
	`</MediaPropertiesLayout>`

Example Failure Response:	`<ApiErrors>`
	`<ApiError>`
	`<ErrorType>Authorization Failure</ErrorType>`
	`<ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`<ErrorData>1234</ErrorData>`
	`</ApiError>`
	`</ApiErrors>`

User—Get Media Properties Layout List

This API allows a user to get a list of all media properties layouts configured on the system, including the default media properties layout. Cisco Finesse Release 12.5(1) onward, this API is deprecated.

URI:	https://<FQDN>/finesse/api/User/<UserId>/MediaPropertiesLayouts
Example URI:	https://finesse1.xyz.com/finesse/api/User/<UserId>/MediaPropertiesLayouts
Security Constraints:	Agents and supervisors can use this API. Any user can get a list of media properties layouts if they are signed in or they provide valid authorization credentials when challenged.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Bad Request 400: Finesse API error (for example, the object does not exist, the object is stale, or violation of DB constraint) 401: Authorization Failure 401: Invalid Authorization User Specified 500: Internal Server Error

Example Response:	`<MediaPropertiesLayouts>`
	`    <MediaPropertiesLayout>`
	`        ... Full MediaPropertiesLayout Object ...`
	`    </MediaPropertiesLayout>`
	`    <MediaPropertiesLayout>`
	`        ... Full MediaPropertiesLayout Object ...`
	`    </MediaPropertiesLayout>`
	`    <MediaPropertiesLayout>`
	`        ... Full MediaPropertiesLayout Object ...`
	`    </MediaPropertiesLayout>`
	`</MediaPropertiesLayouts>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>1234</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

User—Get List of Phone Books

This API allows a user to get a list of phone books and the first list of associated contacts for that user, based on the defined range (1 to 6000). Contacts are retrieved from the global phone books first, followed by the team phone books, up to the maximum limit of 6000. Cisco Finesse Release 12.5(1) onward, this API is deprecated.

For more information about the PhoneBook object, see PhoneBook, on page 297.

URI:	https://<FQDN>/finesse/api/User/<id>/PhoneBooks
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/PhoneBooks
Security Constraints:	Agents and supervisors can use this API. Any user can get a list of their own phone books if they are signed in or they provide valid authorization credentials when challenged.
Additional Headers:	"Range: objects=1-6000" The range of contacts to retrieve.
HTTP Method:	GET
Content Type:	—

Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 206: Partial Content 400: Bad Request (the request body is invalid) 400: Finesse API Error (for example, the object does not exist or the object is stale) 401: Authorization Failure 404: User Not Found 416: Invalid Range Specified. Range must be 1– 6000 objects 500: Internal Server Error

Example Response:	`<PhoneBooks>`
	`   <PhoneBook>`
	`      <name>PhoneBook1</name>`
	`      <type>GLOBAL</type>`
	`      <Contacts>`
	`         <Contact>`
	`            ...Full Contact Object...`
	`         </Contact>`
	`            ...Full Contact Object...`
	`         </Contact>`
	`      </Contacts>`
	`   </PhoneBook>`
	`   <PhoneBook>`
	`      <name>PhoneBook2</name>`
	`      <type>TEAM</type>`
	`      <Contacts>`
	`         <Contact>`
	`            ...Full Contact Object...`
	`         </Contact>`
	`         <Contact>`
	`            ...Full Contact Object...`
	`         </Contact>`
	`      </Contacts>`
	`   </PhoneBook>`
	`</PhoneBooks>`

Example Failure Response:	**Example**
	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Authorization Failure</ErrorType>`
	`    <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`    <ErrorData>1234</ErrorData>`
	`  </ApiError>`
	`</ApiErrors>`
	Example
	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Invalid Input</ErrorType>`
	`    <ErrorData></ErrorData>`
	`    <ErrorMessage>Invalid range header format. Format:`
	`objects=1-6000</ErrorMessage>`
	`  </ApiError>`
	`</ApiErrors>>`
	Example
	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Invalid Input</ErrorType>`
	`    <ErrorData></ErrorData>`
	`    <ErrorMessage>Maximum number of contacts cannot exceed`
	`6000</ErrorMessage>`
	`  </ApiError>`
	`</ApiErrors>`

User—Get List of Workflows

This API allows a user to get a list of workflows and workflow actions assigned to that user. Cisco Finesse Release 12.5(1) onward, this API is deprecated.

For more information about the Workflow object, see Workflow, on page 313.

URI:	https://<FQDN>/finesse/api/User/<id>/Workflows
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Workflows
Security Constraints:	Any user can get their own workflows if they are signed in or they provide valid authorization credentials when challenged.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Bad Request (the request body is invalid) 400: Finesse API Error (for example, the object is stale or there is a violation of database constraints) 401: Authorization Failure 404: Not Found (the resource is not found) 500: Internal Server Error

Example Response:	

```
<Workflows>
   <Workflow>
      <name>google ring pop</name>
      <description> Pops a Google web page when an agent phone
rings</description>
      <TriggerSet>
         <type>SYSTEM</type>
         <name>CALL_ARRIVES</name>
         <triggers>
            <Trigger>
               <Variable>
                  <name>mediaType</name>
                  <node>//Dialog/mediaType</node>
                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_EQUAL</comparator>
               <value>Voice</value>
            </Trigger>
            <Trigger>
               <Variable>
                  <name>callType</name>
                  <node>//Dialog/mediaProperties/callType</node>
                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_IN_LIST</comparator>
               <value>ACT_IN,PREROUTE_ACD_IN,PREROUTE_DIRECT_AGENT,
                TRANSFER,OVERFLOW_IN,OTHER_IN,AGENT_OUT,AGENT_INSIDE,
                OFFERED,CONSULT,CONSULT_OFFERED,CONSULT_CONFERENCE,
                CONFERENCE,TASK_ROUTED_BY_ICM,TASK_ROUTED_BY_
                APPLICATION</value>
            </Trigger>
            <Trigger>
               <Variable>
                  <name>state</name>

<node>//Dialog/participants/Participant/mediaAddress[.=${userExtension}]/../state</node>

                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_IN_LIST</comparator>
               <value>ALERTING,ACTIVE,HELD</value>
            </Trigger>
            <Trigger>
               <Variable>
                  <name>fromAddress</name>
                  <node>//Dialog/fromAddress</node>
                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_NOT_EQUAL</comparator>
               <Variable>
                  <name>userExtension</name>
                  <type>SYSTEM</type>
               </Variable>
            </Trigger>
         </triggers>
      </TriggerSet>
      <ConditionSet>
         <applyMethod>ALL</applyMethod>
         <conditions>
            <Condition>
               <Variable>
                  <name>callVariable1</name>
                  <type>SYSTEM</type>
               </Variable>
               <comparator>CONTAINS</comparator>
               <value>1234</value>
            </Condition>
            <Condition>
               <Variable>
                  <name>user.foo.bar[1]</name>

<node>//Dialog/mediaProperties/callvariables/CallVariable/name[.="user.foo.bar[1]"]/../value</node>

                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_NOT_EMPTY</comparator>
            </Condition>
         </conditions>
      </ConditionSet>
      <workflowActions>
         <WorkflowAction>
            <name>Google ring pop</name>
            <type>BROWSER_POP</type>
            <params>
               <Param>
                  <name>windowName</name>
                  <value>google</value>
               </Param>
               <Param>
                  <name>path</name>

<value>http://www.google.com?a=${CallVariable1}&amp;c=cat&amp;${DNIS}&amp;d=${user.foo.bar[1]}</value>

               </Param>
            </params>
            <actionVariables>
               <ActionVariable>
                  <name>callVariable1</name>
                  <type>SYSTEM</type>
                  <testValue>apple</testValue>
               </ActionVariable>
               <ActionVariable>
                  <name>user.foo.bar[1]</name>

<node>//Dialog/mediaProperties/callvariables/CallVariable/name[.="user.foo.bar[1]"]/../value</node>

                  <type>CUSTOM</type>
                  <testValue>1234</testValue>
               </ActionVariable>
            </actionVariables>
         </WorkflowAction>
         <WorkflowAction>
            <name>My Delay</name>
            <type>DELAY</type>
            <params>
               <Param>
                  <name>time</name>
                  <value>10</value>
               </Param>
            </params>
         </WorkflowAction>
      </workflowActions>
   </Workflow>
</Workflows>
```

Example Failure Response:	`<ApiErrors>`
	  `<ApiError>`
	    `<ErrorType>Unauthorized</ErrorType>`
	    `<ErrorMessage>The user is not authorized to perform this operation</ErrorMessage>`
	  `</ApiError>`
	`</ApiErrors>`

User API Parameters

Parameter	Type	Description	Possible Values	Notes
id	String	The ID of the user.	—	If the user is configured in Unified CCE, size is determined by Unified CCE. If the user is configured in Unified CCX, the size is determined by Unified Communications Manager.
uri	String	The URI to get a new copy of the object.	—	—
roles	Collection	List of roles for this user.	Agent, Supervisor	—
-->role	String	One of the roles assigned to this user.	Agent, Supervisor	—
loginId	String	The login ID of the user.	—	—
loginName	String	The login name of the user.	—	—
state	String	The state for this user.	LOGOUT, NOT_READY, READY, RESERVED, RESERVED_OUTBOUND, RESERVED_OUTBOUND_PREVIEW, TALKING, HOLD, WORK, WORK_READY, UNKNOWN	—

Parameter	Type	Description	Possible Values	Notes
stateChangeTime	String	The time at which the state of the user changed to the current state. The format for this parameter is YYYY-MM-DDThh:MM:ss.SSSZ.	—	This parameter is empty if the time of the state change is not available (if no agent state change notification was received yet).
mediaType	String	The type of media under which the dialog is classified.	—	—
pendingState	String	The state to which the user will transition next.	LOGOUT	For Unified CCX deployments, when an agent is in TALKING state and a Finesse failover or reconnect occurs, this parameter is set to LOGOUT. The pendingState parameter indicates that the agent transitions to LOGOUT state when the call ends.

Parameter	Type	Description	Possible Values	Notes
reasonCodeId	Integer	The database ID for the reason code that indicates why the user is in the current state.	If the user has not selected the reason code, this parameter is empty. Otherwise, the value of this parameter is the database ID for the selected reason code.	The value of the reasonCodeId may be -1 in the following cases: • No reason codes are configured for the category. • The agent has just signed in (transitioned from LOGIN to NOT_READY) • A failover occurred. The agent is in NOT_READY state but Finesse could not recover the reasonCode used before failover.
ReasonCode	Collection	Information about the reason code currently associated with this user.	—	—
-->category	String	The category of the reason code.	NOT_READY, LOGOUT	—
-->uri	String	The full URI for the reason code.	—	—
-->code	Integer	CTI code associated with this reason code.	—	—
-->label	String	The label associated with this reason code.	—	—
-->forAll	Boolean	Whether the reason code is global (true) or non-global (false).	true, false	—
systemCode	Boolean	The reserved status of the reason code	true, false	—
-->id	Integer	The ID of the reason code.	—	—

Parameter	Type	Description	Possible Values	Notes
settings	Collection	The settings for this user.	—	The settings parameter is only present for Unified CCE deployments.
-->wrapUpOnIncoming	String	Indicates whether this user required or allowed to enter wrap-up data on an incoming call.	REQUIRED, OPTIONAL, NOT_ALLOWED, REQUIRED_WITH_WRAP_UP_DATA	This parameter applies only to Unified CCE deployments.
-->wrapUpOnOutgoing	String	Indicates whether this user required or allowed to enter wrap-up data on an outgoing call.	REQUIRED, OPTIONAL, NOT_ALLOWED	This parameter applies only to Unified CCE deployments.
-->deviceSelection	String	Indicates whether the CTI device selection is enabled for the agent. For more information about CTI device selection, see *Device Selection for Shared ACD Line* chapter in *Cisco Finesse Administration Guide* and *Select Active Device* chapter in *Cisco Finesse Agent and Supervisor Desktop User Guide*.	enabled, disabled	The login request must contain activeDeviceId when the device selection is enabled, and when the multiple devices are configured for the extension selected by the agent.
extension	String	The extension that this user is currently using.	—	The extension must exist in Unified Communications Manager. If the user is configured in Unified CCE, size is determined by Unified Communications Manager. If the user is configured in Unified CCX, the size is determined by Unified CCX.

Parameter	Type	Description	Possible Values	Notes
mobileAgent	Collection	Indicates that the user is a mobile agent.	—	This parameter is returned for mobile agents only. Finesse supports mobile agents only in Unified CCE deployments.
-->mode	String	The work mode for the mobile agent	CALL_BY_CALL, NAILED_CONNECTION	This parameter is returned for mobile agents only. Finesse supports mobile agents only in Unified CCE deployments.
-->dialNumber	String	The external number that the system calls to connect to the mobile agent.	—	This parameter is returned for mobile agents only. Finesse supports mobile agents only in Unified CCE deployments. Validated by the Unified Communications Manager dial plan.
firstName	String	The first name of this user.	—	
lastName	String	The last name of this user.	—	—
teamId	String	The ID of the team to which this user belongs.	—	—
teamName	String	The name of the team to which this user belongs.	—	—
skillTargetId	String	Unique identifier for the skill target assigned to the agent in the Unified CCE database. This is supported from Cisco Finesse, Release 12.5(1) ES2 onwards.	—	This parameter applies only to Unified CCE deployments.
dialogs	String	URI to the collection of dialogs that the user is a part of.	—	

Parameter	Type	Description	Possible Values	Notes
teams	Collection	If the user has a role of Supervisor, a list of teams that the user supervises.	—	
-->Team	Collection	Set of information for a team.	—	
--->uri	String	The URI to get a new copy of the Team object.	—	
--->id	String	The ID for the team.	—	
--->name	String	The name of the team.	—	
mediaState	—	The state of the user on a manual outbound call from NOT_READY state.	BUSY, IDLE	This parameter is returned for Team API only and not for User API. This parameter applies only to Unified CCX deployments.
logoutAllMedia	Boolean	Determines if the the logout request is for all media channels (true) or only from the channels configured by the Administrator.	true, false	This parameter applies only to Unified CCE deployments, and is used only when signing out. Administartor can configure the signout channels with the CLI **utils finesse user_signout_channel**
activeDeviceId	String	A unique ID of the active device associated with the extension to which the agent is signed in.	—	This parameter is mandatory in the request payload if multiple devices are associated with the extension, and device selection is enabled for the agent.

Parameter	Type	Description	Possible Values	Notes
Devices	Collection	Information about the list of devices associated with the extension agent has signed in. For more information, see Devices API Parameters, on page 79.	—	—

User API Errors

Status	Error Type	Description
400	Bad Request	The request is malformed or incomplete or the extension provided is invalid.
400	Generic Error	An unaccounted for error occurred. The root cause could not be determined.
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized (for example, the mode for a mobile agent or the state for a user)
400	Invalid State	The requested state change is not allowed (for example, a user in LOGOUT state requests a state change to LOGOUT or a supervisor tries to change an agent's state to something other than READY or LOGOUT).
400	Parameter Missing	The extension, state, or requestedAction is not provided. If signing in a mobile agent, the mode or dialNumber is not provided.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (for example, an agent tries to use an API that only a supervisor or administrator is authorized to use).
401	Invalid Authorization User Specified	The authenticated user tried to make a request for another user.
401	Invalid State	A user tried to change to a state that is not supported in the scenario.
401	Invalid Supervisor	A supervisor tried to change the state of an agent who does not belong to that supervisor's team.

Status	Error Type	Description
404	Not Found	The resource specified is invalid or does not exist.
404	User Not Found	The user ID provided is invalid or is not recongnized. No such user exists in CTI.
500	Internal Server Error	Any runtime exception is caught and responded with this error.
503	Service Unavailable	A dependent service is down (for example, the Cisco Finesse Notification Service or Cisco Finesse Database). Finesse is OUT_OF_SERVICE.

Devices

The Devices object represents the list of devices associated with the given extension and includes information about the deviceId, deviceType, and deviceTypeName. The Devices object is structured as follows:

```
<Devices>
    <Device>
        <deviceId>SEP0019305D8EC1</deviceId>
        <deviceType>30018</deviceType>
        <deviceTypeName>Cisco 7961</deviceTypeName>
    </Device>
    <Device>
        <deviceId>CSFJP5550016</deviceId>
        <deviceType>503</deviceType>
        <deviceTypeName>Cisco Unified Client Services Framework</deviceTypeName>
    </Device>
</Devices>
```

Devices API

Devices—Get List of Devices for Extension

This API allows a user to retrieve the list of devices associated with an extension.

URI:	https://<FQDN>/finesse/api/Devices?extension={ext}
Example URI:	https://finesse1.xyz.com/finesse/api/Devices?extension={1001001}
Security Constraints:	Any user with valid authorization credentials can access the devices for any valid extension.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Bad Request 401: Authorization Failure 404: Not Found 500: Internal Server Error

Example Response:	`<Devices>`
	`    <Device>`
	`        <deviceId>SEP0019305D8EC1</deviceId>`
	`        <deviceType>30018</deviceType>`
	`        <deviceTypeName>Cisco 7961</deviceTypeName>`
	`    </Device>`
	`    <Device>`
	`        <deviceId>CSFJP5550016</deviceId>`
	`        <deviceType>503</deviceType>`
	`        <deviceTypeName>Cisco Unified Client Services Framework</deviceTypeName>`
	`    </Device>`
	`</Devices>`

Example Failure Response:	**For Bad Request:**
	`<ApiError>`
	`    <ErrorType>DEVICES_APIS_NOT_ALLOWED</ErrorType>`
	`   <ErrorMessage>Devices APIs are available only on Finesse servers connected with CTI protocol version 24 or higher</ErrorMessage>`
	`</ApiError>`
	Note This error is applicable only for Unified CCE deployments.
	For Not Found:
	`<ApiError>`
	`    <ErrorType>DEVICES_NOT_FOUND</ErrorType>`
	`    <ErrorMessage>No devices found for the extension 1001001</ErrorMessage>`
	`</ApiError>`

Devices API Parameters

Parameter	Type	Description	Possible Values	Notes
Devices	Collection	Information about the list of devices associated with an extension.	—	—
Device	Collection	Information about a single device.	—	—
-->deviceId	String	A unique ID of the device.	—	—
-->deviceType	Integer	The device type as defined in the CiscoTerminal.getType() in JTAPI specifications.	—	—

Parameter	Type	Description	Possible Values	Notes
-->deviceTypeName	String	The display name of the device type as defined in the CiscoTerminal.getTypeName() in JTAPI specifications. For more information about JTAPI specifications, refer to *Cisco Unified JTAPI Developers Guide*.	—	—

Devices API Errors

Status	Error Type	Description
400	Bad Request	• The request is malformed or incomplete or the extension provided is invalid. • The API is denied as it is applicable only for Cisco Finesse with connected CTI protocol version 24 and above only for Unified CCE deployments.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session).
404	Not Found	If the extension has no devices associated or extension is not valid.
500	Internal Server Error	Any runtime exception is caught and responded with this error. For example, when connection is not established with CTI server or any other component.

Dialog

The Dialog object represents a dialog with participants.

Dialog Object for Voice Calls

For the media type “voice”, this object represents a call. A participant represents an internal or external user's CallConnection, or that user's leg of the call.

The Dialog object is structured as follows for voice calls:

```
<Dialog>
    <associatedDialogUri>/finesse/api/Dialog/321654</associatedDialogUri>
    <id>12345678</id>
    <secondaryId>12345679</secondaryId>
    <mediaType>Voice</mediaType>
    <fromAddress>2002</fromAddress>
    <toAddress>2000</toAddress>
    <mediaProperties>
        <dialedNumber>2000</dialedNumber>
        <callType>AGENT_INSIDE</callType>
        <DNIS>2000</DNIS>
        <queueNumber>5022</queueNumber>
        <queueName>UCM_PIM.Func.Agents.SG</queueName>
        <callKeyCallId>217</callKeyCallId>
        <callKeySequenceNum>1</callKeySequenceNum>
        <callKeyPrefix>152018</callKeyPrefix>
        <wrapUpReason>Sales Call</wrapUpReason>
        <wrapUpItems>
             <wrapUpItem>Wrong number</wrapUpItem>
             <wrapUpItem>Satisfied Customer</wrapUpItem>
        </wrapUpItems>
        <callvariables>
            <CallVariable>
                <name>callVariable1</name>
                <value>Chuck Smith</value>
            </CallVariable>
            <CallVariable>
                <name>callVariable2</name>
                <value>Cisco Systems, Inc.</value>
            </CallVariable>
...Other CallVariables ...
        </callvariables>
    </mediaProperties>
    <participants>
        <Participant>
            <actions>
                <action>HOLD</action>
                <action>DROP</action>
            </actions>
            <mediaAddress>2002</mediaAddress>
            <mediaAddressType>AGENT_DEVICE</mediaAddressType>
            <startTime>2014-02-11T16:10:23.121Z</startTime>
            <state>ACTIVE</state>
            <stateCause></stateCause>
            <stateChangeTime>2014-02-11T16:10:23.121Z</stateChangeTime>
        </Participant>
        <Participant>
            <actions>
                <action>RETRIEVE</action>
                <action>DROP</action>
            </actions>
            <mediaAddress>2000</mediaAddress>
            <mediaAddressType>AGENT_DEVICE</mediaAddressType>
            <startTime>2014-02-11T16:10:23.121Z</startTime>
            <state>HELD</state>
            <stateCause></stateCause>
            <stateChangeTime>2014-02-11T16:10:36.543Z</stateChangeTime>
        </Participant>
    </participants>
    <state>ACTIVE</state>
    <uri>/finesse/api/Dialog/12345678</uri>
    <scheduledCallbackInfo>
        <callbackTime>2014-03-07T14:30</callbackTime>
        <callbackNumber>9785551212</callbackNumber>
    </scheduledCallbackTime>
    <services>
        <service>AgentAnswers</service>
        ... other services ...
    </services>
   <serviceConfigId>AXVKhwtrnNQBVJR2n4uq</serviceConfigId>
   <callGUID>FMEMBLAAAAABAAAAAAAAAAHMFMAKAKAK</callGUID>
</Dialog>
```

Note The <callGUID>, <services> and <serviceConfigId> elements only apply to Unfied CCE deployments.

The <wrapUpItems> element applies only to Unified CCX deployments.

Dialog Object for Nonvoice Tasks

For nonvoice media types, this object represents a task. A participant represents an internal or external user's leg of the task.

The Dialog object is structured as follows for nonvoice tasks:

Note Several Dialog parameters do not apply for nonvoice tasks, and are returned empty.

```
<Dialog>
    <associatedDialogUri>/finesse/api/Dialog/3216_5432_1</associatedDialogUri>
    <id>1234_5423_1</id>
    <mediaType>Cisco_Chat_MRD</mediaType>
    <mediaProperties>
        <mediaId>5002</mediaId>
        <dialedNumber></dialedNumber>
        <queueNumber>5022</queueNumber>
        <queueName>UCM_PIM.Func.Agents.SG</queueName>
        <callKeyCallId>217</callKeyCallId>
        <callKeySequenceNum>1</callKeySequenceNum>
        <callKeyPrefix>152018</callKeyPrefix>
        <wrapUpReason>Sales Call</wrapUpReason>
        <callvariables>
            <CallVariable>
                <name>callVariable1</name>
                <value>Chuck Smith</value>
            </CallVariable>
            <CallVariable>
                <name>callVariable2</name>
                <value>Cisco Systems, Inc.</value>
            </CallVariable>
           ...Other CallVariables ...
        </callvariables>
    </mediaProperties>
    <participants>
        <Participant>
            <actions>
                <action>ACCEPT</action>
            </actions>
            <mediaAddress>1001001</mediaAddress>
            <startTime>2015-11-19T06:04:27.864Z</startTime>
            <state>OFFERED</state>
            <stateChangeTime>2015-11-19T06:04:27.864Z</stateChangeTime>
        </Participant>
    </participants>
    <state>OFFERED</state>
    <uri>/finesse/api/Dialog/1234_5423_1</uri>
</Dialog>
```

Note callKeyCallId, CallKeySequenceNum, and callKeyPrefix parameters apply only to Unified CCE deployments.

Dialog APIs

Note Finesse obtains the dialogId value from the CallID value defined for the calls by the CTI Server. With some call flows, the messaging between Finesse and the CTI Server refers to an updated CallID value. In most cases, the updated CallID value maintains a relationship to the original CallID value, and therefore Finesse maintains the same dialogId value for the duration of the call flows. However, there are some call flows in which the CallID and dialogId change permanently (for example, in a conference). If you require a better understanding of the relationship between the CallID and dialogId values, you can perform some test call flows and view the webservices logs.

Dialog—Get Dialog

This API allows a user to get a copy of a Dialog object.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/12345678
Security Constraints:	Agents and administrators can use this API. Agents can only get their own Dialog object. Administrators can get any Dialog object.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 401: Invalid Authorization 404: Not Found 500: Internal Server Error

Example Response:	

	see block below

```
<Dialog>
   <uri>/finesse/api/Dialog/12345678</uri>
   <mediaType>Voice</mediaType>
   <state>ACTIVE</state>
   <fromAddress>2002</fromAddress>
   <toAddress>2000</toAddress>
   <mediaProperties>
      <mediaId>1</mediaId>
      <dialedNumber>2000</dialedNumber>
      <callType>AGENT_INSIDE</callType>
      <DNIS>2000</DNIS>
      <queueNumber>5022</queueNumber>
      <queueName>UCM_PIM.Func.Agents.SG</queueName>
      <callKeyCallId>217</callKeyCallId>
      <callKeySequenceNum>1</callKeySequenceNum>
      <callKeyPrefix>152018</callKeyPrefix>
      <wrapUpReason>Another satisfied customer</wrapUpReason>
      <wrapUpItems>
             <wrapUpItem>Wrong number</wrapUpItem>
             <wrapUpItem>Satisfied customer</wrapUpItem>
       </wrapUpItems>
      <callbackNumber>14567</callbackNumber>
      <callvariables>
         <CallVariable>
            <name>callVariable1</name>
            <value>Chuck Smith</value>
         </CallVariable>
         <CallVariable>
            <name>callVariable2</name>
            <value>Cisco Systems, Inc</value>
         </CallVariable>
         <CallVariable>
            <name>callVariable3</name>
            <value>chucksmith@cisco.com</value>
         </CallVariable>
         ...Other Call Variables (up to 10)
         <CallVariable>
            <name>ecc.user</name>
            <value>csmith</value>
         </CallVariable>
         <CallVariable>
            <name>ecc.years[0]</name>
            <value>1985</value>
         </CallVariable>
         <CallVariable>
            <name>ecc.years[1]</name>
            <value>1995</value>
         </CallVariable>
   </mediaProperties>
   <participants>
      <Participant>
         <actions>
            <action>HOLD</action>
            <action>DROP</action>
         </actions>
         <mediaAddress>1081001</mediaAddress>
         <mediaAddressType>AGENT_DEVICE<mediaAddressType>
         <startTime>2014-02-04T15:33:16.653Z</startTime>
         <state>ACTIVE</state>
         <stateCause></stateCause>
         <stateChangeTime>2014-02-04T15:33:26.653Z</stateChangeTime>
      </Participant>
      <Participant>
```

	`<actions>`
	`   <action>RETRIEVE</action>`
	`   <action>DROP</action>`
	`</actions>`
	`<mediaAddress>1081002</mediaAddress>`
	`<mediaAddressType>AGENT_DEVICE<mediaAddressType>`
	`<startTime>2014-02-04T15:33:16.653Z</startTime>`
	`<state>HELD</state>`
	`<stateCause></stateCause>`
	`<stateChangeTime>2014-02-04T15:33:27.584Z</stateChangeTime>`
	`</Participant>`
	`</participants>`
	`</Dialog`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Not Found</ErrorType>`
	`      <ErrorMessage>Invalid dialogId specified for dialog/ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Note The <wrapUpItems> element applies only to Unified CCX deployments.

Dialog—Create a New Dialog (Make a Call)

This API allows a user to make a call. To make a call, a new Dialog object is created that specifies the fromAddress (the caller's extension) and the toAddress (the destination target). The new Dialog object is posted to the Dialog collection for that user.

In a Unified CCE deployment, you can also use this API to pass call variables with the MAKE_CALL request. The API supports call variable 1 through call variable 10 and ECC variables. You cannot pass BA variables or wrap-up reasons with the request.

In a Unified CCE Release 12.5(1) deployment onward, you can make a call from Ready state. When the agent goes off-hook to place a call, the Unified CCE changes the agent status to Not Ready with 50006 reason code. The Not Ready state prevents the agent from receiving an inbound call while the outbound call placed by the agent is in progress.

This API supports the use of any ASCII character in the toAddress. Finesse does not convert any entered letters into numbers, nor does it remove non-numeric characters (including parentheses and hyphens) from the toAddress.

Note In a Unified CCX deployment, you cannot use this API to pass call variables. If you supply the mediaProperties parameter with a MAKE_CALL request in a Unified CCX deployment, Finesse returns a 400 Invalid Input error.

URI:	https://<FQDN>/finesse/api/User/<id>/Dialogs
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Dialogs

Security Constraints:	All users can use this API. Users can only create dialogs using a fromAddress to which they are currently signed in. Users with only agent or supervisor permission can make calls.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	`   <requestedAction>MAKE_CALL</requestedAction>`
	`   <fromAddress>1001001</fromAddress>`
	`   <toAddress>1002002</toAddress>`
	`</Dialog>`

HTTP Request with Call Variables (Unified CCE only):	`<Dialog>`
	`   <requestedAction>MAKE_CALL</requestedAction>`
	`   <fromAddress>1001001</fromAddress>`
	`   <toAddress>1002002</toAddress>`
	`   <mediaProperties>`
	`      <callvariables>`
	`         <CallVariable>`
	`            <name>callVariable1</name>`
	`            <value>testcallvar1</value>`
	`         </CallVariable>`
	`         <CallVariable>`
	`            <name>user.Finesse_ecc2</name>`
	`            <value>A</value>`
	`         </CallVariable>`
	`         <CallVariable>`
	`            <name>user.finesse_array[0]</name>`
	`            <value>array_val_0</value>`
	`         </CallVariable>`
	`         <CallVariable>`
	`            <name>user.finesse_array[1]</name>`
	`            <value>array_val_1</value>`
	`         </CallVariable>`
	`         <CallVariable>`
	`            <name>user.finesse_array[2]</name>`
	`            <value>array_val_2</value>`
	`         </CallVariable>`
	`         <CallVariable>`
	`            <name>user.finesse_array[3]</name>`
	`            <value>array_val_3</value>`
	`         </CallVariable>`
	`         <CallVariable>`
	`            <name>user.finesse_array[4]</name>`
	`            <value>array_val_4</value>`
	`         </CallVariable>`
	`      </callvariables>`
	`   </mediaProperties>`
	`</Dialog>`

Request Parameters:	id (required): The ID of the user requested Action (required): The way in which the dialog is created (MAKE_CALL) fromAddress (required): The extension with which the user is currently signed in toAddress (required): The destination for the call mediaProperties (optional): Collection of media-specific properties related to the dialog callvariables (optional): Collection of call variables to include as part of the initial call CallVariable (optional): Name and value pair for a call variable
HTTP Response:	202: Successfully Accepted **Note** This response only indicates successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Bad Request (the request body is invalid) 400: Parameter Missing 400: Invalid Input (a request in a Unified CCX deployment includes mediaProperties) 400: Invalid Destination (the toAddress and fromAddress are the same) 401: Authorization Failure 401: Invalid Authorization 500: Internal Server Error

Example Failure Response:	Authorization error returned when a user has no permission:
	
	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>Unauthorized</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`
	
	Authorization error returned when users with only administrator permission makes a call:
	
	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Invalid Authorization User Specified</ErrorType>`
	`    <ErrorData>The user is not authorized to perform this operation</ErrorData>`
	`    <ErrorMessage>User (administrator) is not an authorizied supervisor.</ErrorMessage>`
	`  </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Invalid State	Attempt to POST a Dialog when the agent is in an invalid state to make a call.	All
Invalid State	Supervisor attempts to POST a Dialog when that supervisor is silently monitoring another agent.	All
Generic Error	Attempt to POST a Dialog to a route point when there are no agents in Ready state in the queue corresponding to that route point.	All
Generic Error	Attempt to POST a Dialog in which the toAddress is an E164 extension.	Unified CCE

Dialog—Take Action on Participant

This API allows a user to take action on a participant within a dialog. Agents must be the participant they are targeting with an action.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	Agents can use this API. Agents can only act on a participant of a dialog when they are that participant.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	**For voice dialogs:**
	`<Dialog>`
	`    <targetMediaAddress>1001001</targetMediaAddress>`
	`    <requestedAction>ANSWER</requestedAction>`
	`</Dialog>`
	voice dialog TRANSFER example:
	`<Dialog>`
	`   <requestedAction>TRANSFER</requestedAction>`
	`   <toAddress>1001002</toAddress>`
	`   <targetMediaAddress>1001001</targetMediaAddress>`
	`</Dialog>`
	voice dialog CONFERENCE example:
	`<Dialog>`
	`   <requestedAction>CONFERENCE</requestedAction>`
	`   <toAddress>1001002</toAddress>`
	`   <targetMediaAddress>1001001</targetMediaAddress>`
	`</Dialog>`
	For nonvoice dialogs:
	Nonvoice dialog CLOSE example:
	`<Dialog>`
	`   <requestedAction>CLOSE</requestedAction>`
	`   <mediaProperties>`
	`       <wrapUpReason>Happy customer!</wrapUpReason>`
	`   </mediaProperties>`
	`</Dialog>`
	Nonvoice dialog TRANSFER example:
	`<Dialog>`
	`   <requestedAction>TRANSFER</requestedAction>`
	`   <target>scriptSelector</target>`
	`</Dialog>`

Request Parameters:	**For voice dialogs:** dialogId (required): The ID of the dialog targetMediaAddress(required): The extension with which the user is currently signed in (used to locate the participant to target with the action request). requestedAction (required): The action to take on the targeted participant **For nonvoice dialogs:** dialogId (required): The ID of the dialog requestedAction (required): The action to take on the targeted participant mediaProperties (optional): A collection of media-specific properties for the dialog. This parameter can be used only when the action is CLOSE in order to set the wrapUpReason parameter. wrapUpReason (optional): A description of the task. This parameter can be used only when the action is CLOSE. target (required for TRANSFER): The Script Selector/dialed number to which the dialog is being transferred.

HTTP Response:	202: Successfully Accepted 400: Parameter Missing (the targetMediaAddress or requestedAction is not provided) 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 404: Dialog Not Found 500: Internal Server Error 503: Service Unavailable (for example, the Notification Service is not running).

Example Failure Response:	**For voice dialogs:**
	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Invalid Input</ErrorType>`
	`          <ErrorData>requestedAction</ErrorData>`
	`          <ErrorMessage>Invalid 'requestedAction' specified for dialog</ErrorMessage>`
	`     </ApiError>`
	`</ApiErrors>`
	For nonvoice dialogs:
	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Agent is not logged in</ErrorType>`
	`          <ErrorMessage>E_ARM_STAT_AGENT_NOT_LOGGED_IN</ErrorMessage>`
	`          <ErrorData>6</ErrorData>`
	`          <ErrorMedia>5001</ErrorMedia>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	**For voice dialogs:** Dialog notification Dialog CTI error notification (if a CTI error occurs) **For nonvoice dialogs:** Dialogs/Media notification Dialogs/Media asynchronous error notifications including CTI errors

Platform-Based API Differences

The following table describes API differences between a stand-alone Finesse deployment with Unified CCE or a coresident Finesse deployment with Unified CCX.

<table>
<tr><th>Scenario</th><th>Response</th></tr>
<tr><td>A participant who is not the conference controller tries to conference in another participant.</td><td><b>Stand-alone Finesse with Unified CCE:</b>
<pre>
<data>
    <apiErrors>
      <apiError>
        <errorData>20999</errorData>
        <errorMessage><ConferenceCallCommand>: Conference
         failed...causes include agent already has a consult call
         or conferencing a non-conference controller.
        </errorMessage>
        <errorType>Generic Error</errorType>
      </apiError>
    </apiErrors>
</data>
</pre>
<b>Coresident Finesse with Unified CCX:</b>
<pre>
<data>
    <apiErrors>
      <apiError>
        <errorData>22</errorData>
        <errorMessage>CF_INVALID_OBJECT_STATE</errorMessage>
        <errorType>Invalid State</errorType>
      </apiError>
    </apiErrors>
</data>
</pre></td></tr>
</table>

Asynchronous Errors for Voice Dialogs

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Generic Error	Attempt a call transfer without an existing consult call.	All
Generic Error	Attempt a call transfer on the original call (a direct call) after the original call has already been retrieved.	All
Generic Error	Attempt to complete a conference on the original call after retrieving the original call.	All
Generic Error	Attempt to exceed the maximum allowed conference participants.	All
Generic Error	Attempt to RETRIEVE an incoming OutBoundPreview campaign call when the allowed actions are ACCEPT, CLOSE, and REJECT.	All
Generic Error	Non-conference-controller attempts to conference in another party.	Unified CCE

ErrorType	Reason	Deployment Type
Generic Error	Attempt to put the held call (a direct call) on hold again.	All
Invalid State	Non-conference-controller attempts to conference in another party.	Unified CCX

Asynchronous Errors for Nonvoice Dialogs

If an error occurs after the initial validation of a nonvoice dialog is complete, the API send an error notification over XMPP to the Dialogs/Media notification. The error message has the format described in "Media and Dialogs/Media Asynchronous Error Notification.". The requestId is included in the response XML. The ErrorMedia parameter in the ApiError information indicates the Media Routing Domain to which the error applies.

For transfers, Finesse communicates asynchronously with Customer Collaboration Platform to initiate task resubmission requests. The following types of errors can occur, and are returned asynchronously:

- Customer Collaboration Platform can respond to the Finesse transfer request with an HTTP error response (for example 4XX or 5XX).
- The Finesse request to Customer Collaboration Platform may time-out due to network issues.

If the request to Customer Collaboration Platform fails, the API send an error notification over XMPP to the Dialogs/Media notification, and Finesse retains the dialog.

Dialog—Update Call Variable Data

This API allows a user to set or change call variables (including named variables or ECC variables) of a dialog. If the user is an agent, the user must be a participant to invoke this action. A corresponding notification is published if there is an update to any of the values of the call variables or named variables.

Note With Unified CCE, Cisco Finesse does not support the use of extended ASCII characters required for additional alphabets in the ECC variables and call variables 1-10. You must use only ASCII characters in the 0-127 range. For example, if you set call variable 2 to contain the character à (ASCII 133), it does not appear correctly on the agent desktop.

With Unified CCX, Cisco Finesse only supports Latin1 characters for ECC variables. Other Unicode characters are not supported. For example, if a user tries to use this API to update an ECC variable that contains Chinese characters, Finesse may not return the correct value in the subsequent dialog update it sends to the client.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	Agents can use this API. Agents can only act on a participant of a dialog when they are that participant.
HTTP Method:	PUT
Content Type:	Application/XML

Input/Output Format:	XML

HTTP Request:	**For voice dialogs**:
	`<Dialog>`
	`   <requestedAction>UPDATE_CALL_DATA</requestedAction>`
	`   <mediaProperties>`
	`      <wrapUpReason>Happy customer!</wrapUpReason>`
	`      <wrapUpItems>`
	`         <wrapUpItem>Wrong number</wrapUpItem>`
	`         <wrapUpItem>Satisfied customer</wrapUpItem>`
	`      </wrapUpItems>`
	`      <callvariables>`
	`         <CallVariable>`
	`            <name>callVariable1</name>`
	`            <value>123456789</value>`
	`         </CallVariable>`
	`         <CallVariable>`
	`         ... Other call variables to be modified ...`
	`         </CallVariable>`
	`      </callvariables>`
	`      </callvariables>`
	`   </mediaProperties>`
	`</Dialog>`
	For nonvoice dialogs:
	`<Dialog>`
	`<requestedAction>UPDATE_CALL_DATA</requestedAction>`
	`<mediaProperties>`
	`<callvariables>`
	`<CallVariable>`
	`<name>{name of the call variable/named variable}</name>`
	`<value>{value to be changed}</value>`
	`</CallVariable>`
	`<CallVariable>`
	`... Other call variables to be modified ...`
	`</CallVariable>`
	`</callvariables>`
	`</mediaProperties>`
	`</Dialog>`

Request Parameters:	**For voice dialogs**: dialogId (required): The ID of the dialog mediaProperties (required): Collection of media-specific properties related to the dialog to be modified wrapUpReason (optional): A description of the call wrapUpItems (required if multiple wrap-up item parameter is present): Contains the list of wrap-up items belonging to this dialog (CCX deployments only). wrapUpItem (optional): A description of the call (CCX deployments only). callvariables (optional): A list of call variables to modify (either wrapUpReason or callvariables must be present in the request) CallVariable (required if the callvariables parameter is present): Contains the name and value of a call variable belonging to this dialog. The name must be present and cannot be empty. Duplicate names cannot exist. The value tag must be specified but can be empty. **For nonvoice dialogs**: dialogid (required) - The ID of the Task requestedAction (required): The action to take
HTTP Response:	202: Successfully Accepted 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 404: Dialog Not Found 500: Internal Server Error 503: Service Unavailable (for example, the Notification Service is not running)

Example Failure Response:	**For voice dialogs**:
	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`
	For nonvoice dialogs:
	`<apiErrors>`
	`    <apiError>`
	`        <errorData>XXX</errorData>`
	`        <errorMedia>5001</errorMedia>`
	`        <errorMessage>XXXXXXXX</errorMessage>`
	`        <errorType>XXXXXXXXXXXXXXX</errorType>`
	`    </apiError>`
	`</apiErrors>`

Notifications Triggered:	Dialog notification Dialog CTI error notification (if a CTI error occurs)

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Invalid Input	The value of a call variable or ECC variable is longer than what is either allowed or configured as the maximum length for that variable.	All
Invalid Input	The value of an ECC variable that is configured as a scalar is set as an array.	All
Invalid Input	The value of an ECC variable that is configured as an array is set as a scalar.	All
Invalid Input	The value of an ECC variable that is configured as an array is set as an array but with an index greater than what is configured.	All
Call Variable is protected	Attempt to set call variables on a non-routed (direct) call.	All

ECC and Call Variable Error Handling

When a client makes an invalid update request for a ECC or call variable, that request is sent to Finesse and then to the CTI server. The CTI server logs certain errors but does not return events for them. In these cases, Finesse does not return an error. Clients must be aware of this behavior and follow the appropriate Unified CCE/Unified CCX documentation.

A client can also send an update request for an ECC or call variable that contains both valid and invalid data (that is, some of the ECC or call variable updates in the request payload are valid while others are invalid). See the following table to determine the response from Finesse in these error scenarios.

Error Scenario	CTI Server Response	Finesse Response
1. A request was sent that generates an error from the CTI server to Finesse. 2. The request payload contained no valid ECC or call variables.	The CTI server sends an error to Finesse.	Finesse forwards the error to the client.
1. A request was sent that generates an error from the CTI server to Finesse.	1. The CTI server sends an error to Finesse.	1. Finesse forwards the error to the client.

2. The request payload contained a mix of valid and invalid ECC or call variables.	**2.** The CTI server does not send an UPDATE_CALL_DATA event to Finesse (that is, the CTI server fails the entire request).	**2.** The client does not receive an UPDATE_CALL_DATA event.
1. A request was sent that does not generate an error from the CTI server to Finesse. **2.** The request payload contained no valid ECC or call variables.	The CTI server does not respond.	Finesse does not respond.
1. A request was sent that does not generate an error from the CTI serverto Finesse. **2.** The request payload contained a mix of valid and invalid ECC or call variables.	**1.** The CTI server does not send an error to Finesse. **2.** The CTI server sends an UPDATE_CALL_DATA event to Finesse for the valid ECC and call variables.	**1.** Finesse does not forward an error to the client. **2.** Finesse forwards the UPDATE_CALL_DATA event to the client.

Note When the size of the value of an ECC variable name exceeds its maximum length, the CTI server silently truncates the value and updates the variable. As a result, Finesse does not receive a maximum length error.

Users of this API must ensure that the variables they are trying to update exist. Users must follow the exact format of each variable and ensure that the maximum size is not exceeded.

Dialog—Send DTMF String

This API allows a user to send a dual-tone multifrequency (DTMF) string during a call.

CTI communication architecture has been optimized in Cisco Finesse Release 12.5(1), which has introduced changes in the Finesse API behavior. As a result of this change, it is suggested that call control requests for the same device should not be sent to the Finesse server until the response to a previous call control request has been received. Multiple DTMF requests can however be send one after another, and the server queues them up for you without any error.

To prevent CTI errors, the Finesse desktop disables **Wrap-Up** button and call control buttons **Hold**, **Transfer**, **Consult**, and **End** across all calls when the DTMF **Keypad** is opened until the responses to all of the DTMF requests have been completed or timed out. It is suggested that third-party clients follow the same design. The number of outstanding DTMF requests and the timeout duration can be configured using the Finesse CLI. For more information on CLIs, see the *Desktop Properties* section in *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	Agents can use this API. An agent must be a participant in the dialog to perform this action.
HTTP Method:	PUT

Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	`   <requestedAction>SEND_DTMF</requestedAction>`
	`   <targetMediaAddress>1001001</targetMediaAddress>`
	`   <actionParams>`
	`      <ActionParam>`
	`         <name>dtmfString</name>`
	`         <value>777</value>`
	`      </ActionParam>`
	`   </actionParams>`
	`</Dialog>`

Request Parameters:	dialogId (required): The ID of the dialog requestedAction (required): The way in which the dialog is created (SEND_DTMF) targetMediaAddress (required): The extension of the agent actionParams (required): A collection of objects called ActionParam, which contain name/value pairs. The name must be dtmfString. The value is the DTMF string to submit and can contain 0-9, *, #, or A-D for Unified CCE. For Unified CCX, the value can only contain 0-9, *, or #.
HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 401: Invalid State (the targetMediaAddress specifies an extension of a participant in HELD state) 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Platform-Based API Differences

The following table describes API differences between a stand-alone Finesse deployment with Unified CCE or a coresident Finesse deployment with Unified CCX.

Scenario	Response
Send a DTMF request with an alphanumeric dtmfString.	**Stand-alone Finesse with Unified CCE:** Unified CCE accepts the alphanumeric dtmfString. **Coresident Finesse with Unified CCX:** Unified CCX allows only 0-9, *, or # in the dtmfString. Using any other values results in the following error: `<apiError> <errorData>3</errorData> <errorMessage>CF_VALUE_OUT_OF_RANGE</errorMessage> <errorType>Generic Error</errorType> </apiError>`

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Generic Error	Attempt to send a DTMF request with a valid requestedAction, a valid targetMediaAddress (agent's extension), and an alphanumeric dtmfString. Unified CCX allows only 0-9, *, and # for the dtmfString. Any other values result in the error.	Unified CCX
Generic Error	Attempt to send a DTMF request for a call when the participant in the dialog whose extension is the targetMediaAddress is in a HELD state.	ALL
Generic Error	Attempt a PUT request to send DTMF while a call is alerting.	ALL

Dialog—Make a Consult Call Request

This API allows an agent to make a consult call request. After the request succeeds, the agent can complete the call as a conference or transfer. The requestedAction for a consult call is CONSULT_CALL. The request is sent to the Dialog URL of an existing active call, from where the call is initiated.

Finesse supports the transfer or conference of any held call to the current active call, as long as the agent performing the transfer or conference is a participant in both the held and active call. Finesse does not support blind conference through the API or the desktop.

Blind conference is defined as follows:

An agent has an active call and initiates a consult call to a destination. The agent starts a conference while the call is ringing at the destination.

Note Only the conference controller (the agent who initiates the conference) can add parties to that conference. For example, Agent 1 is on a call with a customer. Agent 1 consults with Agent 2 and then conferences Agent 2 into the call. Agent 2 then consults with Agent 3. If Agent 2 tries to add Agent 3 to the conference, the request fails.

Finesse maintains a copy of the call variables (including call peripheral variables and ECC variables) for each call in the system. When Unified CCE or Unified CCX sets the call variables to values that are not NULL (through CTI events, such as CALL_DATA_UPDATE_EVENT), the call variables maintained by Finesse are updated with these values. In this way, Finesse ensures that a client always receives the latest data for call variables sent by Unified CCE/Unified CCX. Because an empty string is considered a valid value, when call values are set to empty strings, Finesse updates its version of the same call variables to empty strings and then updates the clients.

Note An agent or supervisor who signs in after being on an active conference call with other devices (which are not associated with any other agent or supervisor) may experience unpredictable behavior with the Finesse Desktop due to incorrect Dialog notification payloads. These limitations also encompass failover scenarios where failover occurs while the agent or supervisor is participating in a conference call. For example, an agent is on a conference call when the Finesse server fails. When that agent is redirected to the other Finesse server, that agent could see unpredictable behavior on the desktop. Examples of unpredictable behavior include, but are not limited to, the following:

- The desktop does not reflect all participants in a conference call.
- The desktop does not reflect that the signed-in agent or supervisor is in an active call.
- Dialog updates contain inconsistent payloads.

Despite these caveats, users may continue to perform usual operations on their phones. Desktop behavior will return to usual after the agent or supervisor drops off the conference call.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	Agents can use this API. An agent must be a participant in the dialog and the agent's extension must match the targetMediaAddress.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	CONSULT_CALL Example
	`<Dialog>`
	`   <requestedAction>CONSULT_CALL</requestedAction>`
	`   <toAddress>1001002</toAddress>`
	`   <targetMediaAddress>1001001</targetMediaAddress>`
	`</Dialog>`

Request Parameters:	dialogId (required): The ID of the dialog requestedAction (required): The way in which the dialog is created (CONSULT_CALL) toAddress (required): The destination for the call targetMediaAddress (required): The extension of the agent, used to locate the participant to target with the requestedAction
HTTP Response:	202: Successfully Accepted 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification Dialog CTI error notification (if a CTI error occurs)

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Generic Error	Attempt a CONSULT_CALL on an incoming OutBoundPreview campaign call while the allowed actions are ACCEPT, CLOSE, and REJECT.	ALL
Generic Error	Attempt a CONSULT_CALL while the call is alerting.	ALL
Generic Error	Attempt a CONSULT_CALL while the call is on HOLD.	ALL

Dialog—Initiate a Single Step Transfer

This API allows a user to make a single-step transfer request. After a user makes a successful request, that user's active call is transferred to the destination provided in the toAddress parameter.

The requestedAction for a single-step transfer is TRANSFER_SST. This request is sent on the Dialog URL of an existing active call, from where the call is initiated. Therefore, the dialogId in the URL represents the dialogId of the active call.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	Agents can use this API. An agent must be a participant in the dialog and the agent's extension must match the targetMediaAddress.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	`   <requestedAction>TRANSFER_SST</requestedAction>`
	`   <toAddress>1001002</toAddress>`
	`   <targetMediaAddress>1001001</targetMediaAddress>`
	`</Dialog>`

Request Parameters:	dialogId (required): The ID of the dialog requestedAction (required): The way in which the dialog is created (TRANSFER_SST) toAddress (required): The destination to which to transfer the call targetMediaAddress (required): The extension of the agent who is making the request
HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 400: Invalid Destination 401: Authorization Failure 401: Invalid Authorization User Specified 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Generic Error	Attempt a TRANSFER_SST before the call gets answered.	Unified CCE
Generic Error	Attempt a TRANSFER_SST on an incoming OutBoundPreview campaign call while the allowed actions are ACCEPT, CLOSE, and REJECT.	Unified CCE

Dialog—Make a Silent Monitor Call

This API allows a supervisor to silently monitor an agent who is on an active call and in TALKING state. A new dialog is created, specifying the fromAddress (the supervisor's extension) and the toAddress (the agent's extension). The dialog is posted to the supervisor's dialog collection.

Note Agent phones to be monitored must support silent monitoring and must be configured in Cisco Unified Communications Manager as follows:

- The correct device type must be configured.
- The device must have Bridge Monitoring enabled.
- The correct permissions must be configured (under User Management > End User > PG User, in the Permissions area, select Standard CTI Allow Call Recording, and then click Add to User Group).

URI:	https://<FQDN>/finesse/api/User/<id>/Dialogs
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Dialogs

Security Constraints:	Supervisors can use this API. A supervisor must be signed in to the fromAddress (extension) being used to create the silent monitor call. Agent to be monitored must be assigned to a team that the supervisor is responsible for. A supervisor can silently monitor any call except a silent monitor call. If an agent drops from or transfers the call that the supervisor is monitoring, the silent monitoring session ends.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	`   <requestedAction>SILENT_MONITOR</requestedAction>`
	`   <fromAddress>1001002</fromAddress>`
	`   <toAddress>1001001</toAddress>`
	`</Dialog>`

Request Parameters:	id (required): The ID of the user requestedAction (required): The way in which the dialog is created (SILENT_MONITOR) fromAddress (required): The extension of the supervisor who initiated the silent monitor request toAddress (required): The extension of the agent that the supervisor wants to monitor
HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 400: Invalid Destination 400: Invalid State 401: Authorization Failure 401: Invalid Authorization User Specified 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Platform-Based API Differences

Stand-alone Finesse with Unified CCE:

In a stand-alone Finesse deployment with Unified CCE, supervisors can silently monitor agents who are on ICD calls or non-ICD calls (for example a call to another agent). The supervisor must be in NOT_READY state to start a silent monitoring session and the agent must be in TALKING state. After the supervisor starts the silent monitoring session, the supervisor transitions to TALKING state.

Coresident Finesse with Unified CCX:

In a coresident Finesse deployment with Unified CCX, supervisors can silently monitor agents who are on ICD calls or non-ICD calls (for example, calls to another agent). The supervisor must be in NOT_READY state to start a silent monitoring state. The agent can be in TALKING state (on an ICD call) or NOT_READY state (on a non-ICD call). After the supervisor starts the silent monitoring call, the supervisor remains in NOT_READY state.

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
88049	Attempt to POST Silent Monitor for an agent who is in Ready, Wrap-Up, Hold, or Not Ready state.	Unified CCX
13145	Attempt to POST Silent Monitor for an agent who is in Hold or Not Ready state.	Unified CCE
Invalid State	Attempt to POST Silent Monitor for an agent who is in Ready or Wrap-Up State.	Unified CCE

Dialog—End a Silent Monitor Call

This API allows a supervisor to drop a silent monitor call that was initiated by that supervisor. The Dialog object is updated by specifying a requestedAction of DROP and the targetMediaAddress of the extension of the supervisor who initiated the silent monitor call.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/32458
Security Constraints:	Supervisors and administrators can use this API. A supervisor can only end a silent monitor call that was initiated by that supervisor. An administrator can end any silent monitor call.

HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	`   <requestedAction>DROP</requestedAction>`
	`   <targetMediaAddress>1001002</targetMediaAddress>`
	`</Dialog>`

Request Parameters:	dialogId (required): The ID of the dialog requestedAction (required): The action to take on the targeted participant (DROP) targetMediaAddress (required): The extension of the supervisor who initiated the silent monitor call
HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 404: Not Found (the dialog specified by the dialogId does not exist) 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Dialog—Make a Barge Call

This API allows a supervisor to barge in to an agent call that the supervisor is silently monitoring. The request specifies the fromAddress (supervisor's extension), the toAddress (agent's extension), and the associatedDialog (the URI of the silent monitor dialog that the supervisor initiated). When the barge request succeeds, the agent's original Dialog object is updated and is posted to the supervisor's dialog collection. The supervisor's silent monitor call is dropped. After the barge request succeeds, the original silent monitor call becomes a conference call with the supervisor, agent, and caller as participants.

The call must meet certain conditions for the barge request to succeed:

- Unified Communications Manager may limit the number of phone devices that can join a conference call (a configurable parameter). When a supervisor makes a barge call, the supervisor is added as a new party to the conference. If the resource limit has already been reached, the supervisor's barge request fails.
- Both Unified CCE and Unified CCX allow a barge request only through the conference controller (the agent who initiates the conference call). In case of CVP routed calls, the barge request is also possible for agents other than the conference controller. If the original call is not a conference call, after the barge request succeeds, the call becomes a conference call and the agent is the conference controller. If the original call is a conference call and the agent is not the conference controller, the supervisor's barge request fails.

URI:	https://<FQDN>/finesse/api/User/<id>/Dialogs
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Dialogs
Security Constraints:	Supervisors can use this API. Supervisors can only make barge call requests using the fromAddress that they are currently signed in to and can only barge in to calls they are already silent monitoring. Administrators cannot barge in to any calls because they are not associated with a phone device.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	`   <requestedAction>BARGE_CALL</requestedAction>`
	`   <fromAddress>1001002</fromAddress>`
	`   <toAddress>1001001</toAddress>`
	
	`<associatedDialogUri>/finesse/api/Dialog/6873122</associatedDialogUri>`
	`</Dialog>`

Request Parameters:	requestedAction (required): The way in which to create the dialog (BARGE_CALL) fromAddress (required): The extension of the supervisor who initiated the barge request toAddress (required): The extension of the agent whose call the supervisor wants to barge in on associatedDialogUri (required): The relative URI of the silent monitor dialog on which the supervisor wants to barge in

HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 400: Invalid Destination 400: Invalid State 400: 20700 (Conference resource limit violation) 400: 20999 (Barge via non-conference-controller or the agent already has an outstanding consult call) 401: Authorization Failure 401: Invalid Authorization User Specified 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Platform-Based API Differences

Stand-alone Finesse with Unified CCE:

A supervisor must be silently monitoring a call before making a request to barge in to that call. In a Finesse deployment with Unified CCE, the supervisor's state during the silent monitoring session is TALKING. When the supervisor barges in to the call, the supervisor's state remains TALKING. The agent's state is TALKING before the silent monitoring request, during the silent monitoring session, and after the barge request succeeds.

Coresident Finesse with Unified CCX:

A supervisor must be silently monitoring a call before making a request to barge into that call. In a coresident Finesse deployment with Unified CCX, the supervisor is in NOT_READY state during the silent monitoring session. If the agent is on an ICD call, the supervisor's state transitions to TALKING after barging in to the call. The agent's state is TALKING before the silent monitoring request, during the silent monitoring session, and after the barge request succeeds.

If the agent is on a non-ICD call (for example, a call to another agent), both the supervisor and the agent remain in NOT_READY state during the silent monitoring session and after the barge request succeeds.

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Generic Error	Supervisor attempts a barge call on an agent who is not the conference controller.	ALL
Generic Error	Supervisor attempts a barge call on an agent who is on a Consult call.	ALL

Dialog—End a Barge Call

This API allows a supervisor to leave a barge call that was initiated by that supervisor. The Dialog object is updated, specifying a requestedAction of DROP and a targetMediaAddress of the extension of the supervisor who made the barge call.

The agent can remain on the call unless the total number of participants becomes less than two when the supervisor leaves (like the drop operation of a conference call).

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/32458
Security Constraints:	Supervisors can use this API. A supervisor can only drop barge call if that supervisor is a participant in the call.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	   `<requestedAction>DROP</requestedAction>`
	   `<targetMediaAddress>1001002</targetMediaAddress>`
	`</Dialog>`

Request Parameters:	requestedAction (required): The way in which to create the dialog (DROP) targetMediaAddress (required): The extension of the supervisor who initiated the barge call

HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 404: Not Found (the dialog specified by the dialogId does not exist) 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Dialog—Drop Participant from Conference

This API allows an agent or supervisor to make a request to drop other participants from a conference based on the permission set by the administrator. By default, this API is only available for supervisors, but the administrator can use the Finesse CLI to expand access to conference controllers or even all agents. This setting is reflected in the **enableDropParticipantFor** property. This API is only supported for Unified CCE deployments.

The possible values for the **enableDropParticipantFor** property are:

- *supervisor_only*—(default value) Only the supervisor, who is a participant of the conference call, can drop other agents in the conference call.
- *conference_controller_and_supervisor*—The supervisor who is a participant of the conference call or an agent who initiated the conference call (conference controller) can drop other participants including CTI Route points and IVR ports.
- *all*—Any agent or supervisor who is a participant of the conference call can drop other participants including CTI Route points and IVR ports.

For example, if the permission is set to be *supervisor_only*, when the supervisor barges in to a call between an agent and a customer, the supervisor is the only one who can make a request to drop the agent from the call, leaving the supervisor on the call with the customer.

For more information, see the *Service Properties* section in *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

After the participant is dropped from the conference, the call may become a two-party call or remain a conference call (if more than two parties remain on the call).

Note When the CLI property is set to a default value (*supervisor_only*), the supervisor can drop only an extension that corresponds to a signed-in agent or supervisor. The supervisor cannot drop a CTI Route Point, IVR port, a device to which no agent is signed in, a caller device, or other agents for whom SILENT_MONITOR is not initiated by the supervisor. In conference calls, the application (CTI softphone) is able to drop only its own connection from the conference and is not able to drop other participants from the conference call.

For more information, see the *Enable Dropping Call Participants from a Conference Call* section in *Cisco Contact Center Gateway Deployment Guide for Cisco Unified ICM/CCE* at https://www.cisco.com/c/en/us/support/customer-collaboration/unified-contact-center-enterprise/products-programming-reference-guides-list.html

If wrap-up is enabled for the agent who is dropped, that agent can perform wrap-up after being dropped.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	The Agents and the supervisor who are the participants in the conference call can use this API. **Note** By default, this API is only available for supervisors, but the administrator can use the Finesse CLI to expand access to conference controllers or even all agents.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	   `<requestedAction>PARTICIPANT_DROP</requestedAction>`
	   `<targetMediaAddress>1001001</targetMediaAddress>`
	`</Dialog>`

Request Parameters:	requestedAction (required): The action to be performed on the dialog. targetMediaAddress (required): The extension to drop from the conference call.

HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 400: Invalid Destination (the targetMediaAddress is not one of the parties in the dialog or is not an agent extension) 400: Invalid State (the dialog is not a conference call) 401: Authorization Failure 401: Invalid Authorization User Specified 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Generic Error	Supervisor barges in and attempts to drop a participant in a two-party call scenario.	ALL

Dialog—Start Recording

This API allows a user to start recording an active call.

Note This API applies to Unified CCX deployments only. If you attempt to use this API on a Finesse deployment with Unified CCE, Finesse returns a “Not Implemented” error.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>

Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	Agents and supervisors can use this API. A user must be a participant in the call to perform this action. An agent cannot record the call of another agent. A supervisor cannot record an agent's call if the supervisor is not a participant in the call. If a supervisor wants to record an agent's call, the supervisor must first start a silent monitoring session on the call. A supervisor can only silently monitor (and therefore record) agents who belong to teams assigned to that supervisor.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	   `<requestedAction>START_RECORDING</requestedAction>`
	   `<targetMediaAddress>1001001</targetMediaAddress>`
	`</Dialog>`

Request Parameters:	requestedAction (required): The way in which to create the dialog (START_RECORDING) targetMediaAddress (required): The extension of the agent whose call to record
HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 401: Invalid State (the targetMediaAddress specifies an extension of a participant in HELD state) 500: Internal Server Error 501: Not Implemented (a recording attempt was made in a Unified CCE deployment)

Example Failure Response:	`<ApiErrors>`
	     `<ApiError>`
	          `<ErrorType>Authorization Failure</ErrorType>`
	          `<ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	          `<ErrorData>jsmith</ErrorData>`
	     `</ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Generic Error	Attempt to PUT a START_RECORDING when the only allowable action is TRANSFER_SST.	Unified CCX
Generic Error	Attempt to PUT a START_RECORDING when the only allowable action is ANSWER.	Unified CCX
Generic Error	Attempt to PUT a START_RECORDING on a Unified CCE deployment type. This API is only supported with Unified CCX deployment type.	Unified CCE

Dialog—Accept, Close, or Reject an Outbound Option Preview Reservation

This API allows a user to accept, close, or reject a reservation in an Outbound Option Preview campaign. Finesse signals an Outbound Option Preview reservation by posting a dialog notification of type OUTBOUND_PREVIEW to the reserved user.

Note This API applies to Unified CCE only. If you attempt to use this API on a Finesse deployment with Unified CCX, Finesse returns a “Not Implemented” error.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	Agents can use this API. An agent must be a participant in the dialog to perform this action.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	   `<requestedAction>{ACCEPT|CLOSE|REJECT}</requestedAction>`
	   `<targetMediaAddress>1001001</targetMediaAddress>`
	`</Dialog>`

Request Parameters:	dialogId (required): The ID of the dialog requestedAction (required): The action to take on the Outbound Option Preview reservation (ACCEPT, CLOSE, or REJECT) targetMediaAddress (required): The extension of the agent
HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 404: Dialog Not Found 500: Internal Server Error 501: Not Implemented

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Generic Error	Attempt to PUT a Dialog object using an action that is not allowed. For example, attempting a HOLD call when allowed actions are ACCEPT, REJECT, and CLOSE.	All

Dialog—Accept, Close, or Reject a Direct Preview Outbound Reservation

This API allows a user to accept, close, or reject an Direct Preview Outbound reservation . Finesse signals a Direct Preview reservation by posting a dialog notification of type OUTBOUND_PREVIEW to the reserved user.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	Agents can use this API. An agent must be a participant in the dialog to perform this action.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	`   <requestedAction>{ACCEPT|CLOSE|REJECT}</requestedAction>`
	`   <targetMediaAddress>1001001</targetMediaAddress>`
	`</Dialog>`

Request Parameters:	dialogId (required): The ID of the dialog requestedAction (required): The action to take on the Direct Preview reservation (ACCEPT, CLOSE, or REJECT) targetMediaAddress (required): The extension of the agent
HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 404: Dialog Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Dialog—Reclassify a Direct Preview Call

This API allows a user to reclassify an Outbound Option Direct Preview call. A call can be reclassified as VOICE, FAX, ANS_MACHINE, INVALID, DO_NOT_CALL, or BUSY. The call type is then sent back to Unified CCX for processing.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321
Security Constraints:	Agents can use this API. Agents can only act on their own Dialog object.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Dialog>`
	`   <requestedAction>RECLASSIFY</requestedAction>`
	`   <targetMediaAddress>1001001</targetMediaAddress>`
	`   <actionParams>`
	`      <ActionParam>`
	`         <name>outboundClassification</name>`
	`         <value>FAX</value>`
	`      </ActionParam>`
	`   </actionParams>`
	`</Dialog>`

Request Parameters:	dialogId (required): The ID of the dialog requestedAction (required): The action to perform (RECLASSIFY) targetMediaAddress (required): The extension of the agent who is making the request actionParams (required): A collection of objects called ActionParam, which contain name/value pairs. The name must be outboundClassification. The value can be VOICE, FAX, ANS_MACHINE, INVALID, DO_NOT_CALL, or BUSY. A single parameter must be specified for the value. Any additional parameters are ignored. **Note** The BUSY parameter is not supported in a Finesse deployment with Unified CCE. If used, it returns an invalid input error.

HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed asynchronously and the state change is sent as part of and updated to the Dialog object. The response is in the BAResponse call variable, which contains the value sent to the CTI server for the reclassify action. No confirmation is returned, other than the value in the BAResponse. 400: Bad Request 400: Finesse API Error (for example, the object does not exist or is stale) 400: Parameter Missing 401: Authorization Failure 401: Invalid Authorization User Specified 404: Dialog Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in "Dialog CTI Error Notification."

ErrorType	Reason	Deployment Type
Generic error	Attempt to reclassify a dialog that is not generated by the outbound campaign.	All

Dialog—Schedule or Cancel a Callback

This API allows a user to schedule or cancel a callback. The dialog action UPDATE_SCHEDULED_CALLBACK is used to schedule or update a callback. The dialog action CANCEL_SCHEDULED_CALLBACK is used to cancel a previously scheduled callback.

URI:	https://<FQDN>/finesse/api/Dialog/<dialogId>
Example URI:	https://finesse1.xyz.com/finesse/api/Dialog/54321

Security Constraints:	Agents can use this API. Agents can only act on their own Dialog object.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request (Update Scheduled Callback):	`<Dialog>`
	`   <requestedAction>UPDATE_SCHEDULED_CALLBACK</requestedAction>`
	`   <targetMediaAddress>1001001</targetMediaAddress>`
	`   <actionParams>`
	`      <ActionParam>`
	`         <name>callbackTime</name>`
	`         <value>2013-12-07T14:30</value>`
	`      </ActionParam>`
	`   </actionParams>`
	`</Dialog>`

HTTP Request (Cancel Scheduled Callback):	`<Dialog>`
	`   <requestedAction>CANCEL_SCHEDULED_CALLBACK</requestedAction>`
	`   <targetMediaAddress>100100</targetMediaAddress>`
	`</Dialog>`

Request Parameters:	dialogId (required): The ID of the dialog requestedAction (required): The action to perform (UPDATE_SCHEDULED_CALLBACK, CANCEL_SCHEDULED_CALLBACK) targetMediaAddress (required): The extension of the agent who is making the request actionParams (required): A collection of objects called ActionParam, which contain name/value pairs. The name must be UPDATE_SCHEDULED_CALLBACK. The value can be callbackTime or callbackNumber. A single parameter must be specified for the value. Any additional parameters are ignored.
HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a dialog notification. 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 404: Dialog Not Found 500: Internal Server Error 501: Not Implemented

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Dialog notification

Dialog API Parameters

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
uri	String	The URI to get a new copy of the object.	—	Yes	Yes	
associatedDialog Uri	String	The URI to a Dialog object that is associated with this Dialog object.	/finesse/api/Dialog/dialogId	Yes	Yes	
secondaryId	Numeric	The call ID value assigned to the secondary call.	—	Yes	No	
mediaType	String	The type of media under which this dialog is classified.	The enterprise name of the Media Routing Domain (MRD).	Yes	Yes	
state	String	The last state of this dialog.	For a list of possible values, see State (Dialog) Parameter Values, on page 131.	Yes	Yes	
fromAddress	String	The calling line ID of the caller.	—	Yes	No	
toAddress	String	The destination for the call.	—	Yes	No	

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
callbackNumber	String	In outbound calls, the customer number received by agent may contain the prefix added by dialer. This value indicates the actual number without any prefix.	—	Yes	No	This is applicable for CCE direct preview outbound calls.
mediaProperties	Collection	A collection of media-specific properties for the dialog.	—	Yes	Yes	
-->mediaId	String	The ID of the MRD.	For voice, this value is always 1.	Yes	Yes	
-->dialedNumber	String	The number dialed.	—	Yes	Yes	This parameter is empty for nonvoice tasks.
queueNumber	Numeric	The queue ID of the call.	—	Yes	Yes	
queueName	String	The queue name of the call.	—	Yes	Yes	
callKeyCallId	Numeric	The unique number of the call routed on a particular day.	—	Yes	Yes	Unified CCE only.
CallKeySequenceNum	Numeric	Represents the call sequence number.	—	Yes	Yes	Unified CCE only.
callKeyPrefix	Numeric	Represents the day when the call is routed.	—	Yes	Yes	Unified CCE only.

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
-->callType	String	The type of call.	ACD_IN, PREROUTE_ACD_IN, PREROUTE_DIRECT_ AGENT, TRANSFER, OTHER_IN, OUT, OVERFLOW_IN, AUTO_OUT, AGENT_OUT, AGENT_INSIDE, ASSIST_CALL, BARGE_IN_CONSULT, CONSULT, CONFERENCE, EMERGENCY, SUPERVISOR_MONITOR, SUPERVISOR_WHISPER, SUPERVISOR_BARGE_IN, SUPERVISOR_INTERCEPT, OFFERED, CONSULT_OFFERED, CONSULT_CONFERENCE, NON_ACD, OUTBOUND, OUTBOUND_PREVIEW, OUTBOUND_CALLBACK, OUTBOUND_CALLBACK_ PREVIEW, OUTBOUND_PERSONAL_ CALLBACK, OUTBOUND_PERSONAL_ CALLBACK_PREVIEW, OUTBOUND_DIRECT_ PREVIEW, OO_RESERVATION_PREDICTIVE, OO_CUSTOMER_IVR, PLAY_AGENT_GREETING, RECORD_AGENT_GREETING, TASK_ROUTED_BY_ICM, TASK_ROUTED_BY_APPLICATION, UNMONITORED, VOICE_CALL_BACK.	Yes	No	
-->DNIS	String	The DNIS provided with the call. For routed calls, the DNIS is the route point.	—	Yes	No	

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
-->wrapUpReason	String	A description of the call.	—	Yes	Yes	The maximum size of this parameter is 39 bytes (which equals 39 US English characters).
wrapUpItems	Collection	A list of multiple wrap-up reasons associated with this dailog.	—	Yes	No	Unified CCX only.
wrapUpItem	String	A description of the call.	—	Yes	No	Unified CCX only.
-->callVariables	Collection	A list of call variables associated with this dialog.	—	Yes	Yes	

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
--->CallVariable	String	Contains the name and value of a call variable belonging to this dialog. The name indicates whether the variable is a call variable or an ECC variable Call variable names start with callVariable#, where # is 1-10. ECC variable names (both scalar and array) are prepended with "user". ECC variable arrays include an index enclosed within square brackets located at the end of the ECC array name (for example, user.myarray[2]). Outbound Option call variables provide additional details about an Outbound Option call.	callvariable1 through callvariable10 ECC variables The following Outbound variables: • BACampaign • BAAccountNumber • BAResponse • BAStatus • BADialedListID • BATimeZone • BABuddyName • BACustomerNumber (Unified CCX only) For information about possible values for BAStatus, see Outbound Call Types and BAStatus, on page 147.	Yes	Yes	Size: • Call variable: 40 bytes • ECC/named variable: Sum of all names, values, and index (if array) must be less than or equal to 2000 bytes. Each ECC variable value cannot exceed the length defined in the CTI server administration user interface.
participants	Collection	A list of all participants (both internal and external) involved in the dialog.	—	Yes	Yes	
-->Participant	Collection	Information about one participant in the dialog.	—	Yes	Yes	

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
--->actions	Collection	A list of actions that are allowed for a participant.	For a list of possible values, see Actions Parameter Values, on page 134.	Yes	Yes	
--->mediaAddress	String	Point of contact for the participant.	Possible values include the extension of an agent or ANI for a caller who are participants in the call. For nonvoice dialogs, the value is the agent's id.	Yes	Yes	
—>mediaAddressType	Collection	The device type specified by the mediaAddress.	AGENT_DEVICE or empty string	Yes	No	

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
--->startTime	String	The UTC time when the participant initiated the call or the first time the participant call state becomes active. Finesse uses the Finesse server timestamp (not the CTI even timestamp) to determine the startTime. A time difference may exist between the Finesse server on side A and side B. Although they are synchronized using an NTP server, a few milliseconds of drift may exist. Therefore, the startTime may be different for a participant if Finesse fails over from side A to side B.	The start time in the format YYYY-MM-DDThh:MM:ss.SSSZ or an empty string	Yes	No	

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
						When an agent signs in with an extension that has an active call, Finesse does not have a call object tracking the call and sets the startTime for this participant as an empty string. If the call does have a participant who is an agent, Finesse can reuse the call object for the extension and the startTime is available For example, if an agent is on a call with a customer and then signs in, Finesse does not have the call object. If the agent is on a call with another agent and then signs in, Finesse can reuse the call object for the extension. In a Unified CCE deployment, Finesse on side B is in standby and keeps track of agent states and calls. When failover occurs, Finesse can recover the startTime for the agent. In a Unified CCX deployment, Finesse on side B does not have the agent state or call information. After failover occurs, Finesse sets

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
						the startTime parameter as an empty string.
--->state	String	The last participant state in a dialog.	For a list of possible values, see State (Participant) Parameter Values, on page 137.	Yes	Yes	
--->stateCause	String	The cause for the last participant state in a dialog.	BUSY, BAD_DESTINATION, SUPERVISOR_BARGE_IN, OTHER	Yes	No	This parameter is usually associated with a FAILED participant state.

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
--->stateChangeTime	String	The UTC time when the participant changed to the current state. Finesse uses the Finesse server timestamp (not the CTI even timestamp) to determine the stateChangeTime. A time difference may exist between the Finesse server on side A and side B. Although they are synchronized using an NTP server, a few milliseconds of drift may exist. Therefore, the stateChangeTime may be different for a participant if Finesse fails over from side A to side B.	The state change time in the format YYYY-MM-DDThh:MM:ss.SSSZ or an empty string	Yes	Yes	When Finesse cannot determine the stateChangeTime, this parameter is an empty string. For example, if a participant is in HELD state and a failover occurs, after failover, Finesse can determine that the participant is in HELD state but cannot determine when the call was put on hold. Therefore, Finesse sets the stateChangeTime parameter to an empty string. In a Unified CCE deployment, Finesse on side B is in standby and keeps track of agent states and calls. When failover occurs, Finesse can recover the stateChangeTime for the agent. In a Unified CCX deployment, Finesse on side B does not have the agent state or call information. After failover occurs, Finesse sets the stateChangeTime parameter as an empty string.

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
scheduledCallbackInfo	Collection	For Outbound Option campaigns, provides information about scheduled callbacks.	—	Yes	No	This parameters is provided only if a callback is scheduled for this dialog.
-->callbackTime	String	The callback time in the format YYYY-MM-DDThh:MM (for example, 2013-12-15T11:45). The time is in the customer's timezone. Optionally, a full ISO-8601 format time string (ex. 2013-12-25T23:59:59.9999999+03:00) can be sent, but everything beyond the minutes, including the time zone, is ignored.	—	Yes	No	This parameter is provided only if a callback time has been set. Value returned in the BAReponse: Callback MMDDYYYY HH:MM (for example, Callback 12072013 14:30)
-->callbackNumber	String	The phone number to call for the callback.	—	Yes	No	This parameter is provided only if a callback number has been set. Value returned in the BAResponse: P#<callbackNumber> (for example, P#9780001)
dispositionCode	String	The reason the dialog ended.	For a list of possible values, see Disposition Code Parameter Values for Nonvoice Tasks, on page 150.	No	Yes	

Parameter	Type	Description	Possible Values	Parameter Provided		Notes
				Voice Calls	Nonvoice Tasks	
callGUID	String	CVP Call GUID or Cisco Call GUID identifies the call and is available only when Unified CVP is part of the call flow. [Extracted from the Protocol Call Reference GUID from the CTI Agent_Pre_Call_Event of CTI version 24]	—	Yes	No	Unified CCE only
services	Collection	List of services available for this dialog.	AgentAnswers, Transcription, and Recording	Yes	No	Unified CCE only
serviceConfigId	String	This field is configured in Unified CCE against a Call Type and is configured on the CCE Admin page. It identifies the specific project to be used when providing the AI services and is required for the Agent Answers functionality.	—	Yes	No	Unified CCE only

State (Dialog) Parameter Values

The following table describes possible values for the state (dialog) parameter for voice dialogs:

Dialog State	Description
INITIATING	Indicates that the phone is off the hook at a device
INITIATED	Indicates that the phone is dialing at the device
ALERTING	Indicates that the call is ringing at a device

Dialog State	Description
ACTIVE	Indicates that the dialog has at least one active participant
FAILED	Indicates that the dialog has failed
DROPPED	Indicates that the dialog has no active participants
ACCEPTED	Indicates the user has accepted the OUTBOUND_PREVIEW dialog

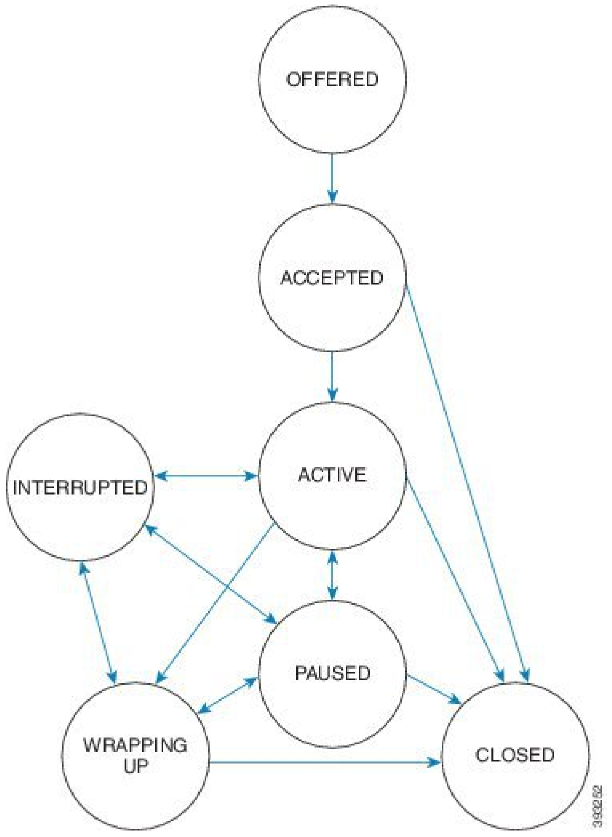
Nonvoice States

The following table describes possible values for the state (dialog) parameter for nonvoice dialogs:

Dialog State	Description
OFFERED	Indicates that the dialog has been offered to a user
ACCEPTED	Indicates that the user has accepted the offered dialog
ACTIVE	Indicates that the dialog has at least one active participant; the user has started working on the accepted dialog
PAUSED	Indicates that an active dialog has been paused
WRAPPING_UP	Indicates that a user is performing wrap up activity for a dialog
INTERRUPTED	Indicates that the dialog has been interrupted by a dialog from another MRD. Dialogs can be interrupted if they are in the ACTIVE, PAUSED, or WRAPPING UP states. While a dialog is interrupted, all actions for that dialog are disabled. This state is applicable only for interruptible MRDs with the Media API interruptAction parameter set to ACCEPT.
CLOSED	Indicates that the dialog ended. The disposition code indicates the reason the dialog closed. See Disposition Code Parameter Values for Nonvoice Tasks, on page 150.

Dialog State	Description
UNKNOWN	After Finesse server or PG failure recovery, any dialogs in the INTERRUPTED state change UNKNOWN state when the dialog is no longer interrupted. For example, the following scenario results in a dialog in the UNKNOWN state: **1.** The user accepts and starts a dialog in an interruptible media. **2.** The user accepts and starts a dialog in a non-interruptible media. **3.** The dialog in the interruptible media changes to the INTERRUPTED state. **4.** The PG goes out of service. **5.** Both dialogs are recovered and are in the correct state. **6.** The user closes the dialog in the non-interruptible media. **7.** The dialog in the interruptible media changes to the UNKNOWN state. When a dialog is in the UNKNOWN state, the user is only allowed to close the dialog.

The following figure illustrates these allowed state transitions for nonvoice dialogs:



Actions Parameter Values

The following table describes possible values (allowable actions) for the Actions response parameter for voice calls:

Participant Allowable Action	**Enabled Button on Desktop**	**Description**
MAKE_CALL	Make a New Call	Allows an agent to make an outgoing call.
ANSWER	Answer	Allows an agent to answer an incoming call.
HOLD	Hold	Allows an agent to hold a call that is currently active.
RETRIEVE	Retrieve	Allows an agent to retrieve a call that was on hold.
DROP	End	Allows an agent to drop the participant of a call.

Participant Allowable Action	Enabled Button on Desktop	Description
UPDATE_CALL_DATA	—	Allows an agent to set call data for the call. **Note** Finesse does not allow an agent to set call data from the desktop. A user can set call data through the API only.
SEND_DTMF	—	Allows an agent to send DTMF digits for the call.
CONSULT_CALL	Consult	Allows an agent to make a consult call for transfer or conference.
CONFERENCE	Conference	Allows an agent to start a conference between the selected held call and the existing active call on the desktop.
TRANSFER	Transfer	Allows an agent to complete a transfer between the selected held call and the existing active call on the desktop.
TRANSFER_SST	Direct Transfer	Allows an agent to initiate a single-step transfer.
SILENT_MONITOR	Start Monitoring	Allows a supervisor to silent monitor an agent who is in TALKING state on an active call.
BARGE_CALL	Barge In	Allows a supervisor to barge in on an agent call that the supervisor is silently monitoring.
PARTICIPANT_DROP	Drop	Allows an agent or a supervisor to drop a participant from a conference call based on the CLI set by the administrator.
UPDATE_SCHEDULED_CALLBACK	Callback, Schedule	Allows an agent to update the details for a scheduled callback.
CANCEL_SCHEDULED_CALLBACK	Callback, Cancel	Allows an agent to cancel a scheduled callback.

Note The Participant Allowable Action is present where applicable for all participants on a call, including participants who are not agents. The actions for participants who are not agents are not needed by the client and may not always be accurate. These actions will be removed in a subsequent release.

Outbound Option Preview Actions

The following table describes the actions available to an agent who is reserved in an Outbound Option Preview campaign, the value to which Finesse sets the BAResponse variable, and the effect it has on the customer number in the campaign.

Note Performing the actions listed in this table causes Finesse to set the BAResponse variable to a corresponding value. Each value triggers a specific action in Unified CCE.

For more information about the BAResponse variable, see the section "Outbound Option Extended Call Variables" in the *Outbound Option Guide for Unified Contact Center Enterprise*.

Action	BAResponse Value	Description
ACCEPT	Accept	Performing the ACCEPT action while reserved in an Outbound Option Preview campaign instructs Unified CCE to establish a call with the customer.
CLOSE	Reject-Close	Performing the CLOSE action while reserved in an Outbound Option Preview campaign rejects the current preview call and prevents the number from being called again in the campaign.
REJECT	Reject	Performing the REJECT action while reserved in an Outbound Option Preview campaign instructs Unified CCE to retry the previewed number later.

Outbound Option Direct Preview Actions

The following table describes the actions available to an agent who is reserved in an Outbound Option Direct Preview campaign, the value to which Finesse sets the BAResponse variable, and the effect it has on the customer number in the campaign.

Note Performing the actions listed in this table causes Finesse to set the BAResponse variable to a corresponding value. Each value triggers a specific action in Unified CCX.

For more information about the BAResponse variable, see the section "Outbound Option Extended Call Variables" in the *Cisco Unified Contact Center Express CTI Protocol Developer Guide*.

Action	BAResponse Value	Description
ACCEPT	Accept	Performing the ACCEPT action while reserved in an Outbound Option Direct Preview campaign instructs Unified CCX to establish a call with the customer.
CLOSE	Reject-Close	Performing the CLOSE action while reserved in an Outbound Option Direct Preview campaign rejects the current preview call and prevents the number from being called again in the campaign.
REJECT	Reject	Performing the REJECT action while reserved in an Outbound Option Direct Preview campaign instructs Unified CCX to retry the previewed number later.
RECLASSIFY	Reclassify	Performing the RECLASSIFY action while reserved in an Outbound Option Direct Preview campaign instructs Unified CCX to reclassify the previewed number as voice (successful case), a modem/fax, answering machine, an invalid number, do not call, or busy.

Nonvoice Actions

The following table describes possible values (allowable actions) for the Actions response parameter for nonvoice tasks:

Action	Description
ACCEPT	Allows an agent to accept an incoming task.
START	Allows an agent to start work on an accepted task.
PAUSE	Allows an agent to pause an active task.
RESUME	Allows an agent to resume a paused task.
TRANSFER	Allows an agent to transfer an accepted, active, or paused task to another Script Selector/dialed number.
WRAP_UP	Allows an agent to perform wrap up work for a task.
CLOSE	Allows an agent to end a task.

State (Participant) Parameter Values

The following table describes possible values for the state (participant) response parameter for voice calls:

Participant State	Allowable Actions for the Participant State	Call State on Finesse Desktop	Description
INITIATING	DROP, UPDATE_CALL_DATA	Off Hook	Indicates that an outgoing call, not yet active, exists on the device
INITIATED	DROP, UPDATE_CALL_DATA	Dialing	Indicates that the phone is dialing at a device
ALERTING	ANSWER	Incoming	Indicates that an incoming call is ringing on the device
ACTIVE	HOLD, DROP, UPDATE_CALL_DATA, CONSULT_CALL	Active	Indicates that the participant is active on the call
FAILED	DROP	Busy	Indicates that the call failed (BUSY)
FAILED	DROP	Error	Indicates that the call failed (BAD_DESINATION)
FAILED	DROP	Error	Indicates that the call failed (OTHER)
HELD	RETRIEVE, DROP, UPDATE_CALL_DATA, TRANSFER (if active call exists), CONFERENCE (if active call exists)	Hold	Indicates that the participant has held their connection to the call
DROPPED	-	-	Indicates that the participant has dropped from the call
WRAP_UP	UPDATE_CALL_DATA	Active	Indicates that the participant is not in active state on the call but is wrapping up after the participant has dropped from the call
ACCEPTED	-	-	Indicates that the participant has accepted the dialog. This state is applicable to OUTBOUND_PREVIEW dialogs.

Note In Finesse Release 9.0(1) and earlier, when a dialog participant wraps up, a dialog event is sent only to the participant who transitions to wrap-up state. In Finesse Release 9.1(1) and later, a dialog event is sent to each participant in the dialog.

Nonvoice State (Participant) Parameter Values

The following table describes possible values (allowable actions) for the Actions response parameter for nonvoice tasks:

Participant State	Allowable Actions for the Participant State	Dialog State	Description
OFFERED	ACCEPT	OFFERED	Indicates that the participant has been offered the task.
ACCEPTED	START, CLOSE, TRANSFER	ACCEPTED	Indicates that the participant has accepted a task, but has not started working on the task.
ACTIVE	PAUSE, WRAP_UP, CLOSE, TRANSFER	ACTIVE	Indicates that the participant is active in the task.
PAUSED	RESUME, CLOSE, TRANSFER, WRAP_UP	PAUSED	Indicates that the participant has paused the active task. The WRAP_UP action is not available if the task was PAUSED from the WRAPPING_UP state.
WRAPPING_UP	PAUSE, CLOSE	WRAPPING_UP	Indicates that the participant is performing wrap up work for a task.
INTERRUPTED	-	INTERRUPTED	Indicates that the participant has been interrupted in this MRD by a task from another MRD. This state is applicable only for interruptible MRDs with the interruptAction parameter set to ACCEPT.
CLOSED	-	-	Indicates that the participant ended the task.

CTI Event Mappings for Dialog and Participant States

The following table provides a list of CTI call events and the associated Dialog and Participant states for the call. This table is specifically oriented toward the agent receiving an incoming call.

Note If the caller is also an agent, the events go to the caller. If the caller is not an agent, events are not published to the caller.

Table 1: Incoming Call

Scenario	CTI Event	Event Method	Dialog State	Participant State (Agent)	Participant State (Caller)
Start the call	BEGIN_CALL_EVENT	POST (Caller)	INITIATING	Not a participant yet	INITIATING
Call arrives at agent	CALL_DELIVERED	POST (Agent), PUT (Caller)	ALERTING	ALERTING	INITIATED
Agent answers call	CALL_ESTABLISHED	PUT	ACTIVE	ACTIVE	ACTIVE
Caller drops call	CALL_CONNECTION_CLEARED	PUT	ACTIVE	ACTIVE	DROPPED
Agent is dropped from call	CALL_CONNECTION_CLEARED	PUT	DROPPED	DROPPED	DROPPED
Call is cleared	CALL_CONNECTION_CLEARED	PUT	DROPPED	DROPPED	DROPPED
Call is removed	END_CALL_EVENT	DELETE	DROPPED	DROPPED	DROPPED

The following table provides a list of CTI call events and their mapping to the Dialog state and Participant state for the call. This table is specifically oriented toward the caller making an outgoing call.

Note If the recipient is also an agent, then the events go to the recipient. If the recipient is not an agent, events are not published to the recipient.

Table 2: Outgoing Call

Scenario	CTI Event	Event Method	Dialog State	Participant State (Caller)	Participant State (Recipient)
Start of any call	BEGIN_CALL_EVENT	POST (Caller)	INITIATING	INITIATING	Not a participant yet

Scenario	CTI Event	Event Method	Dialog State	Participant State (Caller)	Participant State (Recipient)
Caller takes phone off-hook	CALL_SERVICE_INITIATED_EVENT	POST (Caller)	INITIATING	INITIATING	Not a participant yet
Caller dials number	CALL_ORIGINATED_EVENT	PUT (Caller)	INITIATED	INITIATED	Not a participant yet
Destination is busy	CALL_FAILED_EVENT (BUSY)	PUT (Caller)	FAILED	FAILED	Not a participant yet
Destination is bad	CALL_FAILED_EVENT (BAD_DESTINATION)	PUT (Caller)	FAILED	FAILED	Not a participant yet
Destination is recipient	CALL_DELIVERED	PUT (Caller), POST (Recipient) (See the note that precedes this table.)	ALERTING	INITIATED	ALERTING
Recipient answers call	CALL_ESTABLISHED	PUT	ACTIVE	ACTIVE	ACTIVE
Caller drops call	CALL_CONNECTION_CLEARED	PUT	ACTIVE	DROPPED	ACTIVE
Recipient is dropped from call	CALL_CONNECTION_CLEARED	PUT	DROPPED	DROPPED	DROPPED
Call is cleared	CALL_CLEARED_EVENT	PUT	DROPPED	DROPPED	DROPPED
Call is removed	END_CALL_EVENT	DELETE	DROPPED	DROPPED	DROPPED

Note If the caller is also an agent, then the events go to the caller. If the caller is not an agent, events are not published to the caller.

Table 3: Holding a Call

Scenario	CTI Event	Event Method	Dialog State	Participant State (Agent)	Participant State (Caller)
Call arrives and is answered	-	-	-	-	-
Agent holds call	CALL_HELD	PUT	ACTIVE	HELD	ACTIVE
Caller holds call	CALL_HELD	PUT	ACTIVE	HELD	HELD
Agent retrieves call	CALL_RETRIEVED	PUT	ACTIVE	ACTIVE	HELD
Caller retrieves call	CALL_RETRIEVED	PUT	ACTIVE	ACTIVE	ACTIVE

The following table provides a list of CTI call events and their mapping to the Dialog and Participant states for a call transfer. In this scenario, a call exists between the caller and Agent A. The transfer occurs after Agent B answers the consult call.

Table 4: Call Transfer

Scenario	CTI Event (Original Call)	CTI Event (Consult Call)	Event Method	Dialog State	Participant State
Agent A starts consult call	CALL_HELD	-	PUT (original call only)	Original call: ACTIVE	Caller: ACTIVE Agent A: HELD (original call) Agent B: Not yet a participant
Agent A takes phone off-hook (BEGIN_CALL_ EVENT assumed)	-	CALL_SERVICE_ INITIATED_EVENT	PUT (consult call only)	Original call: ACTIVE Consult call: INITIATING	Caller: ACTIVE Agent A: INITIATING (consult call) Agent B: Not yet a participant
Agent A dials number	-	CALL_ORIGINATED_ EVENT	PUT (consult call only)	Original call: ACTIVE Consult call: INITIATED	Caller: ACTIVE Agent A: INITIATED (consult call) Agent B: Not yet a participant

Scenario	CTI Event (Original Call)	CTI Event (Consult Call)	Event Method	Dialog State	Participant State
Agent B receives the call	-	CALL_DELIVERED	PUT (consult call, on Agent A POST (consult call on Agent B	Original call: ACTIVE Consult call: ALERTING	Caller: ACTIVE Agent A: INITIATED (consult call) Agent B: ALERTING
Agent B answers the call	-	CALL_ESTABLISHED	PUT (consult call only)	Original call: ACTIVE Consult call: ACTIVE	Caller: ACTIVE Agent A: ACTIVE (consult call) Agent B: ACTIVE
Agent A completes the transfer of the caller to Agent B	CALL_TRANSFERRED_EVENT	-	DELETE (original call on Agent A) DELETE (consult call on Agent A) DELETE (consult call on Agent B) POST (original call on Agent B)	Original call: DROPPED (Agent A), ACTIVE (Agent B) Consult call: DROPPED (both Agent A and Agent B)	Caller: ACTIVE Agent A: DROPPED (original and consult call) Agent B: DROPPED (consult call), ACTIVE (original call)

If the caller is also an agent, that caller receives a Dialog update (PUT) with an updated participant list after the transfer is complete.

The following table provides a list of CTI call events and their mapping to the Dialog state and Participant state for a silent monitor call.

Note For the Finesse API, a silent monitor call request only specifies the agent's extension for the supervisor to silent monitor. Unified CCE/Unified CCX decides which of the agent's active calls to monitor. In most cases, an agent only has one active call to be monitored. This table describes the scenario where a call already exists between the caller and Agent A. The focus is on the silent monitor call only. In this scenario, the original agent call is not affected. The silent monitor call is created and the agent becomes a participant with no allowable action. The agent has two active calls: the original call and the silent monitor call. Finesse considers the silent monitor call to be a "passive" active call of the agent.

Table 5: Silent Monitor Call

Scenario	CTI Event (Silent Monitor Call)	Event Method	Dialog State (Original Call)	Dialog State (Silent Monitor Call)	Participant State (Caller)	Participant State (Agent A)	Participant State (Supervisor)
Agent call arrives and is answered	-	-	-	-	-	-	-
Supervisor starts the silent monitor call	BEGIN_CALL	POST (SILENT_MONITOR)	ACTIVE	INITIATING	ACTIVE (original call)	ACTIVE (original call)	INITIATING (silent monitor call)
-	CALL_SERVICE_INITIATED_EVENT CALL_DATA_UPDATE_EVENT	-	ACTIVE	INITIATING	ACTIVE (original call)	ACTIVE (original call)	INITIATING (silent monitor call)
-	CALL_ORIGINATED_EVENT CALL_DATA_UPDATE_EVENT	-	ACTIVE	INITIATED	ACTIVE (original call)	ACTIVE (original call)	INITIATED (silent monitor call)
-	CALL_DELIVERED_EVENT CALL_DELIVERED_EVENT	-	ACTIVE	ALERTING	ACTIVE (original call)	ACTIVE (original call)	INITIATED (silent monitor call)

Scenario	CTI Event (Silent Monitor Call)	Event Method	Dialog State (Original Call)	Dialog State (Silent Monitor Call)	Participant State (Caller)	Participant State (Agent A)	Participant State (Supervisor)
-	CALL_ ESTABLISHED_ EVENT	-	ACTIVE	ACTIVE	ACTIVE (original call)	ACTIVE (original call) ACTIVE (passive - silent monitor call)	ACTIVE (silent monitor call)

The following table provides a list of CTI call events and their mapping to the Dialog state and Participant state for a barge call.

Note This table describes a scenario where a call already exists between the caller and Agent A and the supervisor is silently monitoring that call. The focus is on the barge only. In this scenario, the agent call is temporarily put on hold, the silent monitor call is dropped, and a consult call is created. The agent call becomes a conference call with the caller, agent, and supervisor as participants.

Table 6: Barge Call

Scenario	CTI Event	Event Method	Dialog State	Participant State (Caller)	Participant State (Agent A)	Participant State (Supervisor)
Agent call arrives and is answered	-	-	-	-	-	-
Supervisor silent monitors the call	-	-	ACTIVE (original call) ACTIVE (silent monitor call)	ACTIVE	ACTIVE (original call) ACTIVE (passive, silent monitor call)	ACTIVE (silent monitor call)
Supervisor starts barge call	-	POST (BARGE)	ACTIVE (original call) ACTIVE (silent monitor call)	ACTIVE	ACTIVE (original call) ACTIVE (passive, silent monitor call)	ACTIVE (silent monitor call)

Scenario	CTI Event	Event Method	Dialog State	Participant State (Caller)	Participant State (Agent A)	Participant State (Supervisor)
Finesse drops silent monitor call through Unified CCE	CALL_CONNECTION _CLEARED (silent monitor call) CALL_CLEARED (silent monitor call) END_CALL (silent monitor call)	-	ACTIVE (original call) DROPPED (silent monitor call)	ACTIVE (original call)	ACTIVE (original call) ACTIVE (silent monitor call)	DROPPED (silent monitor call)
Unified CCE puts original call on hold	CALL_HELD (original call)	-	ACTIVE (original call)	ACTIVE (original call)	HELD (original call)	Not a participant yet
Unified CCE generates consult call	BEGIN_CALL (consult call) CALL_SERVICE_ INITIATED_EVENT (consult call)	-	ACTIVE (original call) INITIATING (consult call)	ACTIVE	HELD (original call) INITIATING (consult call)	Not a participant yet
Unified CCE dials supervisor's extension	CALL_ORIGINATED_ EVENT (consult call)	-	ACTIVE (original call) INITIATED (consult call)	ACTIVE	HELD (original call) INITIATED (consult call)	Not a participant yet
Agent receives the consult call	CALL_DELIVERED (consult call)	-	ACTIVE (original call) INITIATED (consult call)	ACTIVE	HELD (original call) INITIATED (consult call)	Not a participant yet
Supervisor receives the consult call	CALL_DELIVERED (consult call)	-	ACTIVE (original call) ALERTING (consult call)	ACTIVE	HELD (original call) INITIATED (consult call)	ALERTING
Unified CCE answers the consult call on behalf of the supervisor and changes the original agent call to a conference call	CALL_ CONFERENCED	-	ACTIVE (original call) ALERTING (consult call)	ACTIVE	HELD (original call) INITIATED (consult call)	ALERTING

Scenario	CTI Event	Event Method	Dialog State	Participant State (Caller)	Participant State (Agent A)	Participant State (Supervisor)
Unified CCE ends the consult call	END_CALL (consult call)	-	ACTIVE (original call) DROPPED (consult call)	ACTIVE	HELD (original call) DROPPED (consult call)	-
Unified CCE changes the original call type to conference	CALL_DATA_ UPDATE (original call)	-	ACTIVE (original call)	ACTIVE	ACTIVE (original call, callType=15 =Conference)	-
Unified CCE answers call on behalf of supervisor	CALL_ESTABLISHED (original call)	-	ACTIVE (original call)	ACTIVE	ACTIVE (original call)	ACTIVE

If the caller is also an agent, the caller receives a dialog update (PUT) with an updated participant list on the conference.

Outbound Call Types and BAStatus

The following tables list the call types for outbound calls and the associated values for BAStatus for Unified CCE deployments and Unified CCX deployments.

For each call type, the BAStatus differs based on the outbound agent's mode:

- Dedicated mode—Agents who only make calls for Outbound Option campaigns.
- Blended mode—Agents who receive inbound calls and Outbound Option calls.

Note When a user transfers or conferences an outbound call, the callType changes to TRANSFER or CONFERENCE.

In Unified CCE deployments, the BAStatus of the call remains unchanged. In Unified CCX deployments, the BAStatus changes to TRANSFERRED or CONFERENCED for Progressive and Predictive outbound calls and remains OUTBOUND for Direct Preview outbound calls.

When the failover occurs in a Unified CCE deployment, the callType and BAStatus remain unchanged. In Unified CCX deployments, the callType parameter is null or empty after failover for all outbound dialing modes. The BAStatus parameter is removed as the call no longer functions as an outbound call.

Table 7: Outbound Call Types and BAStatus for Finesse with Unified CCE

	Progressive	Predictive	Preview	Direct Preview
Reservation Call	—	—	callType: OUTBOUND_ PREVIEW BAStatus: BLENDED_PREVIEW_ OUTBOUND_ RESERVATION or PREVIEW_ OUTBOUND_ RESERVATION	callType: OUTBOUND_ DIRECT_ PREVIEW BAStatus: BLENDED_DIRECT_ PREVIEW_ OUTBOUND_ RESERVATION or DIRECT_ PREVIEW_ OUTBOUND_ RESERVATION
Customer Call	callType: OUTBOUND BAStatus: BLENDED_PROGRESSIVE_ OUTBOUND or PROGRESSIVE_ OUTBOUND	callType: OUTBOUND BAStatus: BLENDED_ PREDICTIVE_ OUTBOUND or PREDICTIVE_ OUTBOUND	callType: OUTBOUND BAStatus: BLENDED_PREVIEW_ OUTBOUND or PREVIEW_OUTBOUND	callType: OUTBOUND BAStatus: BLENDED_DIRECT_ PREVIEW_ OUTBOUND or DIRECT_ PREVIEW_ OUTBOUND
Callback Reservation Call	—	—	callType: OUTBOUND_ CALLBACK_PREVIEW BAStatus: PREVIEW_ OUTBOUND_ RESERVATION	callType: OUTBOUND_ DIRECT_ PREVIEW BAStatus: DIRECT_ PREVIEW_ OUTBOUND_ RESERVATION
Callback Customer Call	callType: OUTBOUND_ CALLBACK BAStatus: PROGRESSIVE_ OUTBOUND	callType: OUTBOUND_ CALLBACK BAStatus: PREDICTIVE_ OUTBOUND	callType: OUTBOUND_ CALLBACK BAStatus: PREVIEW_ OUTBOUND	callType: OUTBOUND_ CALLBACK BAStatus: DIRECT_ PREVIEW_ OUTBOUND

	Progressive	Predictive	Preview	Direct Preview
Personal Callback Reservation Call	callType: OUTBOUND_ PERSONAL_ CALLBACK_ PREVIEW BAStatus: PERSONAL_ CALLBACK_ OUTBOUND_ RESERVATION	callType: OUTBOUND_ PERSONAL_ CALLBACK_ PREVIEW BAStatus: PERSONAL_ CALLBACK_ OUTBOUND_ RESERVATION	callType: OUTBOUND_ PERSONAL_ CALLBACK_ PREVIEW BAStatus: PERSONAL_ CALLBACK_ OUTBOUND_ RESERVATION	callType: OUTBOUND_ PERSONAL_ CALLBACK_ PREVIEW BAStatus: PERSONAL_ CALLBACK_ OUTBOUND_ RESERVATION
Personal Callback Customer Call	callType: OUTBOUND_ PERSONAL_ CALLBACK BAStatus: PERSONAL_ CALLBACK_ OUTBOUND	callType: OUTBOUND_ PERSONAL_ CALLBACK BAStatus: PERSONAL_ CALLBACK_ OUTBOUND	callType: OUTBOUND_ PERSONAL_ CALLBACK BAStatus: PERSONAL_ CALLBACK_ OUTBOUND	callType: OUTBOUND_ PERSONAL_ CALLBACK BAStatus: PERSONAL_ CALLBACK_ OUTBOUND

Table 8: Outbound Call Types and BAStatus for Finesse with Unified CCX

	Progressive	Predictive	Direct Preview
Reservation Call	—	—	callType: OUTBOUND_ DIRECT_ PREVIEW BAStatus: DIRECT_ PREVIEW_OUTBOUND_ RESERVATION
Customer Call	callType: OUTBOUND BAStatus: OUTBOUND	callType: OUTBOUND BAStatus: OUTBOUND	callType: OUTBOUND BAStatus: DIRECT_ PREVIEW_ OUTBOUND
Callback Reservation Call	—	—	callType: OUTBOUND_ DIRECT_ PREVIEW BAStatus: DIRECT_ PREVIEW_OUTBOUND_ RESERVATION
Callback Customer Call	callType: OUTBOUND_ CALLBACK BAStatus: OUTBOUND	callType: OUTBOUND_ CALLBACK BAStatus: OUTBOUND	callType: OUTBOUND_ CALLBACK BAStatus: DIRECT_ PREVIEW_ OUTBOUND

	Progressive	Predictive	Direct Preview
Personal Callback Reservation Call	—	—	—
Personal Callback Customer Call	—	—	—

Disposition Code Parameter Values for Nonvoice Tasks

The following table describes possible values for the dispositionCode response parameter for nonvoice tasks:

Type of Code	Disposition Code Value	Description
Normal End	CD_NORMAL_END_TASK	The task ended normally.
Transfer	CD_TASK_TRANSFER	The task was transferred. The initiating application sends a new task request to CCE for routing that includes the task id of the first task.
	CD_TASK_TRANSFERRED_ON_AGENT_LOGOUT	The task was transferred because the agent logged out during the task.
RONA	CD_RING_NO_ANSWER	The task timed out while waiting to be accepted by an agent. The task was redirected to another agent.
Task Lifetime Exceeded	CD_MAX_DIALOG_LIFETIME_EXCEEDED	The dialog ended because it exceeded the maximum task duration for the MRD.
Customer Abandoned	CD_TASK_ABANDONED_WHILE_OFFERED	The customer cancelled the task before the agent began working on the task. In this case, the Finesse user sees the offered dialog but the dialog is deleted before the user can accept it.

Type of Code	Disposition Code Value	Description
Other	CD_CANT_OBTAIN_DIALOG_ID	The Agent PG could not assign an ID to the dialog. In this case, the Finesse user sees the offered dialog, but it is deleted before the user can accept the dialog. Contact Cisco Technical Support for assistance.
	CD_AGENT_LOGGED_OUT_DURING_DIALOG	The agent working on the task logged out before the task ended.
	CD_TASK_ENDED_DURING_APP_INIT	This indicates that the dialog was in progress when the application path went down, and ended before the application path was reinitialized, but within the task life timeout threshold. When the application path was reinitialized, the Agent PG ended the dialog.
	CD_APPLICATION_DISCONNECTED	One instance of an application that is allowed to have multiple client connections with the same application path was disconnected. However, the application path is not down because another instance of the application is still connected.

Dialog API Errors

Status	Error Type	Description
400	20700 (conference resource limit violation)	The barge call will cause the total number of parties on the conference call to exceed the allowed resource limit for the conference bridge.
400	20999 (Barge via a non-conference-controller)	The agent specified in the toAddress is not the controller of the conference call or the agent already has an outstanding conference call.
400	Generic Error	An unaccounted for error occurred. The root cause could not be determined.

Status	Error Type	Description
400	Invalid Destination	The toAddress and fromAddress are the same (if users attempt to call their own extension). For the Dialog—Drop Participant from Conference API, this error occurs if the targetMediaAddress is not one of the parties on the call or is not an agent extension. For the Dialog—Make a Barge Call API, this error occurs if the supervisor tries to barge in on an agent call when the agent's extension is in HELD state.
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized (for example, the fromAddress, toAddress, targetMediaAddress, requestedAction). For the Dialog—Update Call Variable Data API, the call variable name or action is invalid or not recognized, or there are duplicate call variable names. This error is also returned if a user attempts to set any of the following Outbound Option variables: BACampaign, BAAccountNumber, BAResponse, BAStatus, BADialedListID, BATimeZone, BABuddyName, BACustomerNumber (Unified CCX only).
400	Invalid State	A supervisor who is already on an active call (in TALKING or HOLD state) makes a silent monitor request.
400	Parameter Missing	A required parameter was not provided in the request. For example, if creating a dialog, the fromAddess or toAddress was not provided.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (for example, an agent tries to use an API that only a supervisor or administrator is authorized to use).
401	Invalid Authorization User Specified	The authenticated user tried to make a request for another user. The authenticated user tried to use a fromAddress that does not belong to that user.
401	Invalid State	The targetMediaAddress in a Dialog—Start Recording request specifies an extension of a participant in HELD state.

Status	Error Type	Description
401	Invalid Supervisor	A supervisor tried to change the state of an agent who does not belong to that supervisor's team.
404	Not Found	The resource specified is invalid or does not exist.
404	Dialog Not Found	The dialogId provided is invalid or no such dialog exists.
500	Internal Server Error	Any runtime exception is caught and responded with this error.
501	Not Implemented	A user attempted to use the API in a deployment where it is not supported. For example, a recording attempt was made in a Unified CCE deployment.
503	Service Unavailable	The required service is unavailable. For example, the Notification Service is not running.

Queue

The Queue object represents a queue (or skill group in Unified CCE) and contains the URI, name, and statistics for that queue. Queue statistics include the number of calls in queue, the start time of the longest call in queue, and the number of agents in each state.

The Queue object is structured as follows:

```
<Queue>
   <uri>/finesse/api/Queue/10</uri>
   <name>Sales</name>
   <statistics>
      <callsInQueue>3</callsInQueue>
      <startTimeOfLongestCallInQueue>2012-02-15T17:58:21Z</startTimeOfLongestCallInQueue>
      <agentsReady>1</agentsReady>
      <agentsNotReady>2</agentsNotReady>
      <agentsBusyOther>0</agentsBusyOther>
      <agentsLoggedOn>1</agentsLoggedOn>
      <agentsTalkingInbound>3</agentsTalkingInbound>
      <agentsTalkingOutbound>2</agentsTalkingOutbound>
      <agentsTalkingInternal>1</agentsTalkingInternal>
      <agentsWrapUpNotReady>2</agentsWrapUpNotReady>
      <agentsWrapUpReady>3</agentsWrapUpReady>
   </statistics>
</Queue>
```

Queue APIs

Queue—Get Queue

This API allows a user to get a Queue object. Use this API to access statistics for a queue that is assigned to agents or supervisors.

If you use this API to get a queue that is not assigned to any users, the response contains a value of -1 for numeric statistics and is empty for string statistics.

Note This API is only supported for a stand-alone Finesse deployment with Unified CCE and not applicable for coresident Finesse deployment with Unified CCX.

URI:	https://<FQDN>/finesse/api/Queue/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/Queue/10
Security Constraints:	Any user can use this API to retrieve information about a specific queue. The user does not need to belong to that queue.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 404: Not Found 500: Internal Server Error

Example Response:	`<Queue>`
	`    <uri>/finesse/api/Queue/10</uri>`
	`    <name>Sales</name>`
	`    <statistics>`
	`       <callsInQueue>3</callsInQueue>`
	
	`<startTimeOfLongestCallInQueue>2012-02-15T17:58:21Z</startTimeOfLongestCallInQueue>`
	
	`       <agentsReady>1</agentsReady>`
	`       <agentsNotReady>2</agentsNotReady>`
	`       <agentsBusyOther>0</agentsBusyOther>`
	`       <agentsLoggedOn>1</agentsLoggedOn>`
	`       <agentsTalkingInbound>3</agentsTalkingInbound>`
	`       <agentsTalkingOutbound>4</agentsTalkingOutbound>`
	`       <agentsTalkingInternal>5</agentsTalkingInternal>`
	`       <agentsWrapUpNotReady>6</agentsWrapUpNotReady>`
	`       <agentsWrapUpReady>7</agentsWrapUpReady>`
	`    </statistics>`
	`</Queue>`

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

Platform-Based API Differences

The following statistics fields are updated only for a stand-alone Finesse deployment with Unified CCE:

- callsInQueue
- startTimeOfLongestCallInQueue
- agentsReady
- agentsNotReady
- agentsTalkingInbound
- agentsTalkingOutbound
- agentsTalkingInternal
- agentsWrapUpNotReady
- agentsWrapUpReady
- agentsLoggedOn
- agentsBusyOther

Queue—Get List of Queues for User

This API allows a user to get a list of all queues associated with that user. If the user is a supervisor, it returns all queues assigned to all team members of the supervised teams, in addition to the teams that the supervisor belongs to.

Note The list of queues does not include the system-defined queue (skill group) present in Unified CCE to which all agents belong.

URI:	https://<FQDN>/finesse/api/User/<id>/Queues
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Queues
Security Constraints:	All users can use this API to retrieve a list of queues for any user.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 404: User Not Found 500: Internal Server Error

Example Response:	
	```
	<Queues>
	    <Queue>
	        <uri>/finesse/api/Queue/1234</uri>
	        <name>Sales</name>
	        <statistics>
	            <callsInQueue>3</callsInQueue>

	<startTimeOfLongestCallInQueue>2012-02-15T17:58:21Z</startTimeOfLongestCallInQueue>

	            <agentsReady>1</agentsReady>
	            <agentsNotReady>2</agentsNotReady>
	            <agentsBusyOther>0</agentsBusyOther>
	            <agentsLoggedOn>1</agentsLoggedOn>
	            <agentsTalkingInbound>3</agentsTalkingInbound>
	            <agentsTalkingOutbound>4</agentsTalkingOutbound>
	            <agentsTalkingInternal>5</agentsTalkingInternal>
	            <agentsWrapUpNotReady>6</agentsWrapUpNotReady>
	            <agentsWrapUpReady>7</agentsWrapUpReady>
	        </statistics>
	    </Queue>
	    ... more queues ...
	</Queues>
	```

Example Failure Response:	
	```
	<ApiErrors>
	     <ApiError>
	          <ErrorType>Authorization Failure</ErrorType>
	          <ErrorMessage>UNAUTHORIZED</ErrorMessage>
	          <ErrorData>jsmith</ErrorData>
	     </ApiError>
	</ApiErrors>
	```

Platform-Based API Differences

The following statistics fields are updated only for a stand-alone Finesse deployment with Unified CCE:

- callsInQueue
- startTimeOfLongestCallInQueue
- agentsReady
- agentsNotReady
- agentsTalkingInbound
- agentsTalkingOutbound
- agentsTalkingInternal
- agentsWrapUpNotReady
- agentsWrapUpReady
- agentsLoggedOn
- agentsBusyOther

Queue API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the Queue object.	—	
id	String	A unique identifier for the queue. This identifier is the PeripheralNumber from t_Skill_Group in AWDB.	—	
name	String	The name of the queue.	—	
statistics	Collection	A list of statistics for the queue.	—	
-->callsInQueue	Integer	The number of calls currently queued to this queue.	—	If the queue is not assigned to an agent or supervisor, this value is -1.
-->startTimeOf LongestCallInQueue	String	The start time of the longest call in the queue. The format for this parameter is YYYY-MM-DDThh:MM:ssZ.	—	If the queue is not assigned to an agent or supervisor, this value is -1.
-->agentsReady	Integer	The number of agents assigned to the queue who are in READY state.	—	If the queue is not assigned to an agent or supervisor, this value is -1.
-->agentsNotReady	Integer	The number of agents assigned to the queue who are in NOT_READY state.	—	If the queue is not assigned to an agent or supervisor, this value is -1.
-->agentsTalking Inbound	Integer	The number of agents assigned to the queue who are in TALKING state on inbound calls.	—	If the queue is not assigned to an agent or supervisor, this value is -1.

Parameter	Type	Description	Possible Values	Notes
-->agentsTalking Outbound	Integer	The number of agents assigned to the queue who are in TALKING state on outbound calls.	—	If the queue is not assigned to an agent or supervisor, this value is -1. Outbound calls include non-routed calls placed to external devices that are not monitored by Unified Communications Manager or to devices in a different Unified Communications Manager cluster. Outbound Dialer calls are not included.
-->agentsTalking Internal	Integer	The number of agents assigned to the queue who are in Talking state on internal calls. Internal calls are consult calls. When an agent on a routed call initiates an internal consult call, this statistic is incremented for the queue associated with the original call.	—	If the queue is not assigned to an agent or supervisor, this value is -1.
-->agentsWrapUp NotReady	Integer	The number of agents assigned to the queue who are in Work Not Ready state.	—	If the queue is not assigned to an agent or supervisor, this value is -1.
-->agentsWrapUp Ready	Integer	The number of agents assigned to the queue who are in Work Ready state.	—	If the queue is not assigned to an agent or supervisor, this value is -1.
-->agentsBusyOther	Integer	Number of agents currently busy with calls.	—	If the queue is not assigned to an agent or supervisor, this value is -1.

Parameter	Type	Description	Possible Values	Notes
–>agentsLoggedOn	Integer	Number of agents who are currently logged in to the system.	—	If the queue is not assigned to an agent or supervisor, this value is -1.

Queue API Errors

Status	Error Type	Description
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session).
404	Not Found	The resource specified is invalid or does not exist.
404	User Not Found	The user ID provided is invalid or is not recongnized. No such user exists in CTI.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

Team

The Team object represents a team and contains the URI, team name, and the users associated with the team.

The Team object does not contain a full User object for each of the team's users, but a summary object that contains the User uri, loginId, firstName, lastName, ReasonCode, and extension parameters. For more information about these parameters, see *User API Parameters*.

The Team object is structured as follows:

```
<Team>
   <uri>/finesse/api/Team/34</uri>
   <id>34</id>
   <name>My Team</name>
   <users>
      <User>
         <uri>/finesse/api/User/1234/</uri>
         <loginId>1234</loginId>
         <firstName>Charles</firstName>
         <lastName>Brown</lastName>
         <dialogs>/finesse/api/User/1234/Dialogs</dialogs>
         <extension>1001001</extension>
         <pendingState></pendingState>
         <state>LOGOUT</state>
         <stateChangeTime>2012-03-01T17:58:21.345Z</stateChangeTime>
      </User>
      <User>
         <uri>/finesse/api/User/1235/</uri>
         <loginId>1235</loginId>
         <firstName>Jack</firstName>
         <lastName>Brawn</lastName>
         <dialogs>/finesse/api/User/1235/Dialogs</dialogs>
         <extension>1001002</extension>
```

```
                <pendingState></pendingState>
                <state>NOT_READY</state>
                <reasonCode>
                   <category>NOT_READY</category>
                   <code>12</code>
                   <label>Lunch Break</label>
                   <id>1</id>
                   <uri>/finesse/api/ReasonCode/1</uri>
                </reasonCode>
                <stateChangeTime>2012-03-01T18:22:25.123Z</stateChangeTime>
             </User>
             ...Other Users...
          </users>
       </Team>
```

Team APIs

Team—Get Team

This API allows a user to get a copy of the Team object. The Team object contains the configuration information for a specific team, which includes the URI, the team ID, the team name, and a list of agents who are members of that team.

The URI for this API contains the parameter includeLoggedOutAgents. This parameter is optional and can be set to:

- **True** or **Empty**: Includes all the agents of that team in the list (with the logged out agents).
- **False**: Includes only the logged in agents in the list.

URI:	https://<FQDN>/finesse/api/Team/<id>?includeLoggedOutAgents=true
Example URI:	https://finesse1.xyz.com/finesse/api/Team/10?includeLoggedOutAgents=true
Security Constraints:	By default, only administrators and supervisors can access this API. Supervisors can access the information of the teams that they are asigned to and Administrators can access all the teams. If the **enableTeamAPIAccessForAllusers** is enabled, all supervisors and agents can access this API and there will be no restriction. That is, all users can access information of all the teams. For more information about **enableTeamAPIAccessForAllusers**, see the Cisco Finesse Administration Guide.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
Request Parameters:	id (required): The ID of the user includeLoggedOutAgents (optional): Returns the list with all the agents in that team

HTTP Response:	200: Success 401: Authorization Failure 404: Not Found 500: Internal Server Error

Example Response for Unified CCE deployment:	`<Team>`
	`   <uri>/finesse/api/Team/34</uri>`
	`   <id>34</id>`
	`   <name>My Team</name>`
	`   <users>`
	`      <User>`
	`         <uri>/finesse/api/User/1234/</uri>`
	`         <loginId>1234</loginId>`
	`         <firstName>Charles</firstName>`
	`         <lastName>Brown</lastName>`
	`         <dialogs>/finesse/api/User/1234/Dialogs</dialogs>`
	`         <extension>1001001</extension>`
	`         <pendingState></pendingState>`
	`         <state>LOGOUT</state>`
	`         <stateChangeTime>2012-03-01T17:58:21.345Z</stateChangeTime>`
	`      </User>`
	`      <User>`
	`         <uri>/finesse/api/User/1235/</uri>`
	`         <loginId>1235</loginId>`
	`         <firstName>Jack</firstName>`
	`         <lastName>Brawn</lastName>`
	`         <dialogs>/finesse/api/User/1235/Dialogs</dialogs>`
	`         <extension>1001002</extension>`
	`         <pendingState></pendingState>`
	`         <state>NOT_READY</state>`
	`         <reasonCode>`
	`            <category>NOT_READY</category>`
	`            <code>12</code>`
	`            <label>Lunch Break</label>`
	`            <id>1</id>`
	`            <uri>/finesse/api/ReasonCode/1</uri>`
	`         </reasonCode>`
	`         <stateChangeTime>2012-03-01T18:22:25.123Z</stateChangeTime>`
	`      </User>`
	`      ...Other Users...`
	`   </users>`
	`</Team>`

Example Response for Unified CCX deployment:	`<Team>`
	`   <uri>/finesse/api/Team/34</uri>`
	`   <id>34</id>`
	`   <name>My Team</name>`
	`   <users>`
	`      <User>`
	`         <uri>/finesse/api/User/1234/</uri>`
	`         <loginId>1234</loginId>`
	`         <firstName>Charles</firstName>`
	`         <lastName>Brown</lastName>`
	`         <mediaState>BUSY</mediaState>`
	`         <dialogs>/finesse/api/User/1234/Dialogs</dialogs>`
	`         <extension>1001001</extension>`
	`         <pendingState></pendingState>`
	`         <state>LOGOUT</state>`
	`         <stateChangeTime>2012-03-01T17:58:21.345Z</stateChangeTime>`
	`      </User>`
	`      <User>`
	`         <uri>/finesse/api/User/1235/</uri>`
	`         <loginId>1235</loginId>`
	`         <firstName>Jack</firstName>`
	`         <lastName>Brawn</lastName>`
	`         <dialogs>/finesse/api/User/1235/Dialogs</dialogs>`
	`         <extension>1001002</extension>`
	`         <pendingState></pendingState>`
	`         <state>NOT_READY</state>`
	`         <reasonCode>`
	`            <category>NOT_READY</category>`
	`            <code>12</code>`
	`            <label>Lunch Break</label>`
	`            <id>1</id>`
	`            <uri>/finesse/api/ReasonCode/1</uri>`
	`         </reasonCode>`
	`         <stateChangeTime>2012-03-01T18:22:25.123Z</stateChangeTime>`
	`      </User>`
	`      ...Other Users...`
	`   </users>`
	`</Team>`

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

Team—Get List of TeamMessages

This API allows the user to get a list of all active TeamMessages for a particular team.

URI:	https://<FQDN>/finesse/api/Team/<teamid>/TeamMessages
Example URI:	https://finesse1.xyz.com/finesse/api/Team/5000/TeamMessages
Security Constraints:	Agents and Supervisors of the team can use this API.
HTTP Method:	GET
Content Type:	—

Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 404: Not Found 500: Internal Server Error

Example Response:

```
<TeamMessages>
    <TeamMessage>

<uri>/finesse/api/BroadcastMessage/be1598bb-bb2a-4dfc-8c01-91ec10b029af</uri>

        <id>be1598bb-bb2a-4dfc-8c01-91ec10b029af</id>
        <createdBy>
            <id>1001050</id>
            <firstName>AGENT</firstName>
            <lastName>1001050</lastName>
        </createdBy>
        <createdAt>1537418173</createdAt>
        <duration>100</duration>
        <content>content 4</content>
        <teams>
            <team>5052</team>
            <team>5000</team>
        </teams>
    </TeamMessage>
    <TeamMessage>

<uri>/finesse/api/TeamMessage/c652fb4f-1f1a-48c8-bc77-2cbab3c9d231</uri>

        <id>c652fb4f-1f1a-48c8-bc77-2cbab3c9d231</id>
        <createdBy>
            <id>1001050</id>
            <firstName>AGENT</firstName>
            <lastName>1001050</lastName>
        </createdBy>
        <createdAt>1537418172</createdAt>
        <duration>100</duration>
        <content>content 4</content>
        <teams>
            <team>5052</team>
            <team>5000</team>
        </teams>
    </TeamMessage>
    <TeamMessage>

<uri>/finesse/api/TeamMessage/ea74a0db-efcf-4651-84b1-1d2119509e9f</uri>

        <id>ea74a0db-efcf-4651-84b1-1d2119509e9f</id>
        <createdBy>
            <id>1001050</id>
            <firstName>AGENT</firstName>
            <lastName>1001050</lastName>
        </createdBy>
        <createdAt>1537418177</createdAt>
        <duration>100</duration>
        <content>some content 4</content>
        <teams>
            <team>5052</team>
            <team>5000</team>
        </teams>
    </TeamMessage>
</broadcastMessages>
```

Example Failure Response:

```
<ApiErrors>
    <ApiError>
        <ErrorType>Not Found</ErrorType>
        <ErrorData>finesse.api.not_found</ErrorData>
        <ErrorMessage>Team not found.</ErrorMessage>
    </ApiError>
</ApiErrors>
```

Team API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the Team object.	—	
id	String	The unique identifier for the team.	—	
name	String	The name of the team.	—	
users	Collection	The list of users that belong to this team.	—	
-->User	Collection	Information about one specific user on the team.	—	The Team object contains a subset of the User parameters. These parameters include the uri, loginId, firstName, lastName, dialogs, pendingState, state, stateChangeTime, extension, ReasonCode, and mediaState. For information about these parameters, see *User API Parameters*.

Team API Errors

Status	Error Type	Description
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session).
404	Not Found	The team id is invalid. No such team exists.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

TeamResource

The TeamResource object represents a team configuration based on Team assignments. The object contains the URI, team ID, and the respective configuration. The agent or supervisor uses the TeamResource APIs to get copy of the details such as reason codes, wrap-up reasons, media properties layout, phone books, and workflows associated to the team.

From Cisco Finesse Release 12.5(1) onwards, the following User APIs are deprecated. These APIs are available for backward compatibility and have lower performance compared to the TeamResouce APIs.

- User—Get Reason Code List
- User—Get Wrap-Up Reason List
- User—Get Media Properties Layout List
- User—Get List of Phone Books
- User—Get List of Workflows

Responses from the TeamResource APIs are valid for all members of the team.

For more details, see Cisco Finesse REST APIs, on page 4.

TeamResource APIs

TeamResource—Get Reason Codes

This API allows an agent or supervisor to get the NOT_READY, LOGOUT, and ALL reason codes for a team.

Note The ReasonCodes can be empty (for example, if no reason codes for the specified category exist in the Finesse configuration database).

Reason codes that have **forAll** field set to **`true`** apply to all the teams.

The category parameter is required when making a request to get reason codes for a team.

For more information about the ReasonCode object, see ReasonCode, on page 260.

URI:	https://<FQDN>/finesse/api/TeamResource/<teamId>/ReasonCodes?category=NOT_READY|LOGOUT|ALL
Example URI:	https://finesse1.xyz.com/finesse/api/TeamResource/1234/ReasonCodes?category=NOT_READY
Security Constraints:	Agents and supervisors who are part of the team can use this API. To get the reason codes for the team, the user must be signed in or provide valid authorization credentials.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML

HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request (the request body is invalid) 400: Finesse API Error (for example, the object does not exist, the object is stale, or violation of the DB constraint) 401: Authorization Failure 401: Invalid Authorization 404: Not Found (for example, the teamId does not exist or has been deleted) 500: Internal Server Error

Example Response:	`<ReasonCodes category="NOT_READY">`
	`    <ReasonCode>`
	`        <uri>/finesse/api/ReasonCode/1234</uri>`
	`        <category>NOT_READY</category>`
	`        <code>12</code>`
	`        <label>Lunch</label>`
	`        <forAll>true</forAll>`
	`    </ReasonCode>`
	`    <ReasonCode>`
	`      ...Full ReasonCode Object...`
	`   </ReasonCode>`
	`    <ReasonCode>`
	`      ...Full ReasonCode Object...`
	`   </ReasonCode>`
	`</ReasonCodes>`

Example Failure Response:	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Authorization Failure</ErrorType>`
	`    <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`    <ErrorData>1234</ErrorData>`
	`  </ApiError>`
	`</ApiErrors>`

TeamResource—Get Wrap-Up Reasons

This API allows an agent or supervisor to get all the wrap-up reasons for a team.

For more information about the WrapUpReason object, see WrapUpReason, on page 267.

URI:	https://<FQDN>/finesse/api/TeamResource/<teamId>/WrapUpReasons
Example URI:	https://finesse1.xyz.com/finesse/api/TeamResource/1234/WrapUpReasons
Security Constraints:	Agents and supervisors who are part of the team can use this API. To get the wrap-up reason for the team, the user must be signed in or provide valid authorization credentials.
HTTP Method:	GET

Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request (the request body is invalid) 400: Finesse API Error (for example, the object does not exist, the object is stale, or violation of the DB constraint) 401: Authorization Failure 401: Invalid Authorization 404: Not Found (for example, the teamId does not exist or has been deleted) 500: Internal Server Error

Example Response:	`<WrapUpReasons>`
	`  <WrapUpReason>`
	`    <label>Successful tech support call</label>`
	`    <forAll>true</forAll>`
	`    <uri>/finesse/api/WrapUpReason/1234</uri>`
	`  </WrapUpReason>`
	`    ... more wrap-up reasons ...`
	`</WrapUpReasons>`

Example Failure Response:	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Authorization Failure</ErrorType>`
	`    <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`    <ErrorData>1234</ErrorData>`
	`  </ApiError>`
	`</ApiErrors>`

TeamResource—Get Media Properties Layouts

This API allows an agent or supervisor to get the media properties layout configured for the team.

URI:	https://<FQDN>/finesse/api/TeamResource/<teamId>/MediaPropertiesLayouts
Example URI:	https://finesse1.xyz.com/finesse/api/TeamResource/1234/MediaPropertiesLayouts
Security Constraints:	Agents and supervisors who are part of the team can use this API. To get the media properties layout for the team, the user must be signed in or provide valid authorization credentials.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Bad Request (the request body is invalid) 400: Finesse API error (for example, the object does not exist, the object is stale, or violation of the DB constraint) 401: Authorization Failure 401: Invalid Authorization 404: Not Found (for example, the teamId does not exist or has been deleted) 500: Internal Server Error

Example Response:	`<MediaPropertiesLayouts>`
	`    <MediaPropertiesLayout>`
	`        ... Full MediaPropertiesLayout Object ...`
	`    </MediaPropertiesLayout>`
	`    <MediaPropertiesLayout>`
	`        ... Full MediaPropertiesLayout Object ...`
	`    </MediaPropertiesLayout>`
	`    <MediaPropertiesLayout>`
	`        ... Full MediaPropertiesLayout Object ...`
	`    </MediaPropertiesLayout>`
	`</MediaPropertiesLayouts>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>1234</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

TeamResource—Get Phone Books

This API allows an agent or supervisor to get phone books and the first associated contacts for the team, based on the defined range (1 to 6000). Contacts are retrieved from the global phone books first, followed by the team phone books, up to the maximum limit of 6000. For more information about the PhoneBook object, see PhoneBook, on page 297.

URI:	https://<FQDN>/finesse/api/TeamResource/<teamId>/PhoneBooks
Example URI:	https://finesse1.xyz.com/finesse/api/TeamResource/1234/PhoneBooks
Security Constraints:	Agents and supervisors who are part of the team can use this API. To get the phone book for the team, the user must be signed in or provide valid authorization credentials.
Additional Headers:	"Range: objects=16000" The range of contacts to retrieve.
HTTP Method:	GET
Content Type:	—

Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 206: Partial Content 400: Bad Request (the request body is invalid) 400: Finesse API error (for example, the object does not exist, the object is stale, or violation of the DB constraint) 401: Authorization Failure 401: Invalid Authorization 404: Not Found (for example, the teamId does not exist or has been deleted) 416: Invalid Range Specified. Range must be 1– 6000 objects 500: Internal Server Error
Example Response:	See below

```
<PhoneBooks>
   <PhoneBook>
      <name>PhoneBook1</name>
      <type>GLOBAL</type>
      <Contacts>
         <Contact>
            ...Full Contact Object...
         </Contact>
            ...Full Contact Object...
         </Contact>
      </Contacts>
   </PhoneBook>
   <PhoneBook>
      <name>PhoneBook2</name>
      <type>TEAM</type>
      <Contacts>
         <Contact>
            ...Full Contact Object...
         </Contact>
         <Contact>
            ...Full Contact Object...
         </Contact>
      </Contacts>
   </PhoneBook>
</PhoneBooks>
```

Example Failure Response:	**Example**
	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Authorization Failure</ErrorType>`
	`    <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`    <ErrorData>1234</ErrorData>`
	`  </ApiError>`
	`</ApiErrors>`
	Example
	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Invalid Input</ErrorType>`
	`    <ErrorData></ErrorData>`
	`    <ErrorMessage>Invalid range header format. Format:`
	`objects=1-6000</ErrorMessage>`
	`  </ApiError>`
	`</ApiErrors>>`
	Example
	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorType>Invalid Input</ErrorType>`
	`    <ErrorData></ErrorData>`
	`    <ErrorMessage>Maximum number of contacts cannot exceed`
	`6000</ErrorMessage>`
	`  </ApiError>`
	`</ApiErrors>`

TeamResource—Get Workflows

This API allows an agent or supervisor to get the workflows and workflow actions that are assigned to a team.

For more information about the Workflow object, see Workflow, on page 313.

URI:	https://<FQDN>/finesse/api/TeamResource/<teamId>/Workflows
Example URI:	https://finesse1.xyz.com/finesse/api/TeamResource/1234/Workflows
Security Constraints:	Agents and supervisors who are part of the team can use this API. To get the workflows for the team, the user must be signed in or provide valid authorization credentials.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Bad Request (the request body is invalid) 400: Finesse API Error (for example, the object is stale or there is a violation of database constraints) 401: Authorization Failure 401: Invalid Authorization 404: Not Found (for example, the teamId does not exist or has been deleted) 500: Internal Server Error

Example Response:	

```
<Workflows>
   <Workflow>
      <name>google ring pop</name>
      <description> Pops a Google web page when an agent phone
rings</description>
      <TriggerSet>
         <type>SYSTEM</type>
         <name>CALL_ARRIVES</name>
         <triggers>
            <Trigger>
               <Variable>
                  <name>mediaType</name>
                  <node>//Dialog/mediaType</node>
                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_EQUAL</comparator>
               <value>Voice</value>
            </Trigger>
            <Trigger>
               <Variable>
                  <name>callType</name>
                  <node>//Dialog/mediaProperties/callType</node>
                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_IN_LIST</comparator>
               <value>ACT_IN,PREROUTE_ACD_IN,PREROUTE_DIRECT_AGENT,
                TRANSFER,OVERFLOW_IN,OTHER_IN,AGENT_OUT,AGENT_INSIDE,
                OFFERED,CONSULT,CONSULT_OFFERED,CONSULT_CONFERENCE,
                CONFERENCE,TASK_ROUTED_BY_ICM,TASK_ROUTED_BY_
                APPLICATION</value>
            </Trigger>
            <Trigger>
               <Variable>
                  <name>state</name>

<node>//Dialog/participants/Participant/mediaAddress[.=${teamresourceExtension}]/../state</node>

                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_IN_LIST</comparator>
               <value>ALERTING,ACTIVE,HELD</value>
            </Trigger>
            <Trigger>
               <Variable>
                  <name>fromAddress</name>
                  <node>//Dialog/fromAddress</node>
                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_NOT_EQUAL</comparator>
               <Variable>
                  <name>teamresourceExtension</name>
                  <type>SYSTEM</type>
               </Variable>
            </Trigger>
         </triggers>
      </TriggerSet>
      <ConditionSet>
         <applyMethod>ALL</applyMethod>
         <conditions>
            <Condition>
               <Variable>
                  <name>callVariable1</name>
                  <type>SYSTEM</type>
               </Variable>
               <comparator>CONTAINS</comparator>
               <value>1234</value>
            </Condition>
            <Condition>
               <Variable>
                  <name>teamresource.foo.bar[1]</name>

<node>//Dialog/mediaProperties/callvariables/CallVariable/name[.="teamresource.foo.bar[1]"]/../value</node>

                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_NOT_EMPTY</comparator>
            </Condition>
         </conditions>
      </ConditionSet>
      <workflowActions>
         <WorkflowAction>
            <name>Google ring pop</name>
            <type>BROWSER_POP</type>
            <params>
               <Param>
                  <name>windowName</name>
                  <value>google</value>
               </Param>
               <Param>
                  <name>path</name>

<value>http://www.google.com?a=${CallVariable1}&amp;c=cat&amp;${DNIS}&amp;d=${teamresource.foo.bar[1]}</value>

               </Param>
            </params>
            <actionVariables>
               <ActionVariable>
                  <name>callVariable1</name>
                  <type>SYSTEM</type>
                  <testValue>apple</testValue>
               </ActionVariable>
               <ActionVariable>
                  <name>teamresource.foo.bar[1]</name>

<node>//Dialog/mediaProperties/callvariables/CallVariable/name[.="teamresource.foo.bar[1]"]/../value</node>

                  <type>CUSTOM</type>
                  <testValue>1234</testValue>
               </ActionVariable>
            </actionVariables>
         </WorkflowAction>
         <WorkflowAction>
            <name>My Delay</name>
            <type>DELAY</type>
            <params>
               <Param>
                  <name>time</name>
                  <value>10</value>
               </Param>
            </params>
         </WorkflowAction>
      </workflowActions>
   </Workflow>
</Workflows>
```

Example Failure Response:	`<ApiErrors>` `<ApiError>` `<ErrorType>Unauthorized</ErrorType>` `<ErrorMessage>The team resource is not authorized to perform this operation</ErrorMessage>` `</ApiError>` `</ApiErrors>`

TeamResource—Get Layout

Returns the team's desktop layout information of an agent or supervisor requesting it. The users must be part of a team and must request the layout according to their role (agent or supervisor) in the team.

Note The layout information is cached in the reverse-proxy and web-proxy and are refreshed at regular time intervals. If the cached layout information is available users can request for layout information for which they don't have permission (given that the user is part of the same team). For example, an agent can request the layout information of the supervisor role for the same team ID.

URI:	https://<FQDN>/finesse/api/TeamResource/<teamID>/Layout?role=agent|supervisor&finesseLayout=false|true
Example URI:	https://finesse1.xyz.com/finesse/api/TeamResource/1234/Layout?role=agent&finesseLayout=false
Security Constraints:	Agents and supervisors who are part of a team can use this API. To get the layout for a team, users must be signed in or provide valid authorization credentials.
HTTP Method:	GET
Content Type:	—
Input Format	text/plain
Output Format:	JSON—Success Response XML—Error Response
HTTP Request:	`curl --location --request GET 'https://finesse25.autobot.cvp:8445/finesse/api/TeamResource/5000/Layout/?finesseLayout=false&role=agent' \ --header 'Authorization: Basic MTAwMTAwMjpjaXNjbw=='`
HTTP Response:	200: Success 400: Bad Request (the request body is invalid) 401: Invalid Authorization 500: Internal Server Error 501: Not implemented (if query param finesseLayout is true)

Example Response:	

```
{
   "assets": {
   "url": null
   },
   "configs": [
   {
      "key": "title",
      "value": "Cisco Finesse"
   },
   {
      "key": "enableDropParticipantFor",
      "value": "all"
   },
   {
      "key": "dropParticipant",
      "value": "all"
   }
   ],
   "header": {
      "leftAlignedColumns": [
      {
         "gadget": null,
         "component": [
         {
            "attributes": {
            "id": "cd-logo",
            "order": "1"
            },
            "url": "/desktop/scripts/js/logo.js",
            "stylesheet": null
         }
         ],
         "width": "300px"
      }
      ],
   "rightAlignedColumns": [
   {
      "gadget": null,
      "component": [
      {
         "attributes": {
            "id": "broadcastmessagepopover",
            "order": "4"
         },
         "url": "/desktop/scripts/js/teammessage.component.js",
         "stylesheet": null
      }
   ],
   "width": "50px"
   }
   ]
},
"page": {
   "navstype": "overlay",
   "contentarea": {
      "rows": [
      {
         "columns": [
         {
            "gadget": null,
            "component": [
            {
               "attributes": {
                  "id": "alert-banner",
                  "order": "8"
               },
               "url": "/desktop/scripts/js/alertmanager.component.js",
               "stylesheet": null
            }
            ],
            "width": null
         }
         ],
         "height": null
      }
   ],
   "height": null
   },
      "navs": [
      {
         "url": "#/home",
         "deferLoad": null,
         "label": "finesse.container.tabs.agent.homeLabel",
         "icon": "home",
         "iconUrl": null,
         "contentarea": {
            "rows": [
            {
               "columns": [
               {
                  "gadget": null,
                  "component": null,
                  "width": null
               }
               ],
               "height": null
            }
            ],
         "height": null
         },
      "navs": null
      }
      ]
   },
   "footer": null
}
```

Example Unauthorized Error Response

```
<?xml version="1.0" encoding="UTF-8"?>
<ApiErrors>
    <ApiError>
        <ErrorData>Authorization user specified is invalid</ErrorData>
        <ErrorType>Invalid Authorization User Specified</ErrorType>
        <ErrorMessage>HTTP Status code:   401 (Unauthorized)
Api Error Type: Invalid Authorization User Specified
Error Message:  The user specified in the authentication credentials and
 the uri don't match</ErrorMessage>
    </ApiError>
</ApiErrors>
```

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>Invalid role </ErrorType>`
	`        <ErrorData>supervisor1</ErrorData>`
	`        <ErrorMessage>Exception while validating role</ErrorMessage>`
	`    </ApiError>`
	`</ApiErrors>`

TeamResource API Parameters

Parameter	Type	Description	Possible Values	Notes
ReasonCodes	Collection	Information about the reason codes that are currently associated with this team.	—	—
-->category	String	The category of the reason code.	NOT_READY, LOGOUT, ALL	—
-->uri	String	The full URI for the reason code.	—	—
-->code	Integer	Numeric code associated with this reason code.	—	—
-->forAll	Boolean	Whether the reason code is global (true) or non-global (false).	true, false	—
-->systemCode	Boolean	The reserved status of the reason code	true, false	
-->label	String	The label associated with this reason code.	—	—
WrapUpReasons	String	Information about the wrap-up reasons currently associated with this team.	—	—
-->uri	String	The URI to get a new copy of the WrapUpReason object.	—	—
-->label	String	The UI label for the wrap-up reason.	—	Maximum of 39 bytes (which is equal to 39 US English characters). The label must be unique.

Parameter	Type	Description	Possible Values	Notes
-->forAll	Boolean	Whether the wrap-up reason is global (true) or non-global (false).	true, false	
MediaPropertiesLayouts	Collection	Information about the media properties layouts that are currently associated with this team.	—	—
PhoneBooks	Collection	Information about the phone books currently associated with this team.	—	—
name	String	The name of the phone book.	—	—
type	String	The type of phone book.	GLOBAL, TEAM	—
Workflows	Collection	Information about the workflows that are currently associated with this team.	—	For more information on Workflow parameters, see Workflow API Parameters, on page 326.
teamid	String	Team ID for which layout is requested.	—	—
role	String	Agent or Supervisor for which layout is requested.	—	—
fineseLayout	Boolean	Layout information of the requested user.	true, false	For now, the default value is false. If true, the api returns 501 HTTP Status code.

TeamResource API Errors

Status	Error Type	Description
400	Bad Request	The request is malformed or incomplete or the extension that is provided is invalid.
400	Generic Error	An unaccounted error occurred. The root cause could not be determined.

Status	Error Type	Description
400	Invalid Input	One of the parameters provided as part of the input is invalid or not recognized (for example, the state for a team)
400	Invalid State	The requested state change is not allowed (for example, a team in LOGOUT state requests a state change to LOGOUT or a supervisor tries to change an agent's state to something other than READY or LOGOUT).
400	Parameter Missing	The extension, state, or requestedAction is not provided.
401	Authorization Failure	Unauthorized (for example, the team is not yet authenticated in the Web Session).
401	Invalid Authorization	The authenticated team tried to make a request for another team.
401	Invalid State	Team tried to change to the state that is not supported in the scenario.
404	Not Found	The team that is specified is invalid or does not exist.
404	TeamId Not Found	The team details provided is invalid or is not recognized. No such team exists in CTI.
416	Range Not Satisfiable	The range that is specified is invalid or does not exist. For example, the maximum number of contacts cannot exceed 6000.
500	Internal Server Error	Any run-time exception is caught and responded with this error.
503	Service Unavailable	The dependent service is down (for example, the Cisco Finesse Notification Service or Cisco Finesse Database). Finesse is OUT_OF_SERVICE.

Get Script Selectors

This API allows you to get all the available script selectors. You can get the script selectors based on the MRD type (**mrdType**). The "**mrdType**" can be `voice`, `digitalchannel`, or `all`. If you do not specify any "**mrdType**", the `all` option is considered.

URI:	https://<FQDN>/finesse/api/ScriptSelectors?mrdType=<parameter>
Example URI:	https://finesse1.xyz.com/finesse/api/ScriptSelectors?mrdType=all
Security Constraints:	Administrators, agents, and supervisors can use this API.

HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 404: Not Found 500: Internal Server Error

Example HTTP Response:	

mrdType is `all`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ScriptSelectors>
  <ScriptSelector>
    <MrdID>1</MrdID>
    <Name>CVP_VRU_RC1.7000200000</Name>
    <DialedNumber>7000200000</DialedNumber>
    <Description>SIP, Direct, CVPCS20A Dialed Number</Description>

  </ScriptSelector>
  <ScriptSelector>
    <MrdID>1</MrdID>
    <Name>CVP_VRU_RC1.7000202001</Name>
    <DialedNumber>7000202001</DialedNumber>
  </ScriptSelector>
  ...
  <ScriptSelector>
    <MrdID>1</MrdID>
    <Name>MR_PIM1_Voice.PersonalCallback</Name>
    <DialedNumber>PersonalCallback</DialedNumber>
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>5000</MrdID>
    <Name>MR_PIM1_Voice.mark_test_dn</Name>
    <DialedNumber>mark_test_dn</DialedNumber>
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>5000</MrdID>
    <Name>mrsim.mark_test_dn</Name>
    <DialedNumber>mark_test_dn</DialedNumber>
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>5003</MrdID>
    <Name>MR_PIM1_Voice.uq_script_dn</Name>
    <DialedNumber>uq_script_dn</DialedNumber>
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>5003</MrdID>
    <Name>mrsim.uq_script_dn</Name>
    <DialedNumber>uq_script_dn</DialedNumber>
  </ScriptSelector>
</ScriptSelectors>
```

mrdType is `voice`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ScriptSelectors>
  <ScriptSelector>
    <MrdID>1</MrdID>
    <Name>CVP_VRU_RC1.7000200000</Name>
    <DialedNumber>7000200000</DialedNumber>
    <Description>SIP, Direct, CVPCS20A Dialed Number</Description>

  </ScriptSelector>
  <ScriptSelector>
    <MrdID>1</MrdID>
    <Name>CVP_VRU_RC1.7000202001</Name>
    <DialedNumber>7000202001</DialedNumber>
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>1</MrdID>
    <Name>CVP_VRU_RC1.7000202002</Name>
    <DialedNumber>7000202002</DialedNumber>
  </ScriptSelector>
  ...
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>1</MrdID>
    <Name>UCM_RC.218022</Name>
    <DialedNumber>218022</DialedNumber>
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>1</MrdID>
    <Name>MR_PIM1_Voice.PersonalCallback</Name>
    <DialedNumber>PersonalCallback</DialedNumber>
  </ScriptSelector>
</ScriptSelectors>
```

mrdType is `digitalChannels`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ScriptSelectors>
  <ScriptSelector>
    <MrdID>5000</MrdID>
    <Name>MR_PIM1_Voice.mark_test_dn</Name>
    <DialedNumber>mark_test_dn</DialedNumber>
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>5000</MrdID>
    <Name>mrsim.mark_test_dn</Name>
    <DialedNumber>mark_test_dn</DialedNumber>
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>5003</MrdID>
    <Name>MR_PIM1_Voice.uq_script_dn</Name>
    <DialedNumber>uq_script_dn</DialedNumber>
  </ScriptSelector>
  <ScriptSelector>
    <MrdID>5003</MrdID>
    <Name>mrsim.uq_script_dn</Name>
    <DialedNumber>uq_script_dn</DialedNumber>
  </ScriptSelector>
</ScriptSelectors>
```

Example Failure Response:

```
<ApiErrors>
   <ApiError>
      <ErrorType>Unauthorized</ErrorType>
      <ErrorMessage>Not authorized to access this
resource.</ErrorMessage>
   </ApiError>
</ApiErrors>
```

Table 9: Field Details

Field	Description
MrdID	A unique media routing domain (MRD) ID to map with digital routing media channels.
Name	The name of the media channel configured in Unified CCE.

Field	Description
DialedNumber	Dialed numbers, also called as script selectors, are the strings or numbers submitted with Task Routing task requests through Customer Collaboration Platform. Each dialed number is associated with a call type, and determines which routing script the Unified CCE uses to route the request to an agent.
Description	Additional information about the dialed number.

ClientLog

The ClientLog object is a container element that holds client log data to post to the Finesse server. This object supports a POST operation only.

The ClientLog object is structured as follows:

```
<ClientLog>
   <logData>
      ...client logs...
   </logData>
</ClientLog>
```

ClientLog APIs

ClientLog—Post to Finesse

This API is backward compatible with earlier versions of Finesse, it allows a user to submit client-side logs to the Finesse server. Cisco Finesse Release 12.5(1) onwards, use the CompressedClientLog—Post Compressed Log to Finesse, on page 188

URI:	https://<FQDN>/finesse/api/User/<id>/ClientLog
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/ClientLog
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<ClientLog>`
	`    <logData>`
	`        xxxxxxxxxxxxxxx\n`
	`        xxxxxxxxxxxxxxx\n`
	`    </logData>`
	`</ClientLog>`

Request Parameters:	id (required): The ID of the user logData (required): The log data that the client sends to the server

HTTP Response:	202: Successfully Accepted **Note** This response only indicates a successful completion of the request. The request is processed and the actual response is sent as part of a CLIENT_LOG_EVENT that contains empty data elements and a matching requestId. 400: Parameter Missing 400: Invalid Input 400: Operation Failure 401: Authorization Failure 401: Invalid Authorization User Specified 405: Method Not Available

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>User Not Found</ErrorType>`
	`          <ErrorMessage>UNKNOWN_USER</ErrorMessage>`
	`          <ErrorData>4023</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

CompressedClientLog—Post Compressed Log to Finesse

This API allows a user to submit compressed logs to the Cisco Finesse server. The server saves the compressed data in a zip file format.

URI:	https://<FQDN>/finesse/api/User/<id>/CompressedClientLog
Example URI:	https://finesse1.xyz.com/finesse/api/ User/1234/CompressedClientLog
HTTP Method:	POST
Content Type:	multipart/form-data **Note** When you send zip files, select Content Type as form-data.
Input/Output Format:	Binary/XML
HTTP Request:	HTTP Clients have to construct HTTP multipart requests to upload a zip file. The zip file size cannot exceed 1 megabyte.
HTTP Response:	201: Created 400: Bad Request 401: Authorization Failure
Example Response:	—

Example Failure Response:	
	```
	400 (BAD REQUEST)
	<ApiErrors>
	    <ApiError>
	        <ErrorType>Invalid Input</ErrorType>
	        <ErrorData>LOG_DATA</ErrorData>
	        <ErrorMessage>Compressed File Size exceeds max allowed
	size</ErrorMessage>
	    </ApiError>
	</ApiErrors>

	401(Unauthorized)
	<ApiErrors>
	    <ApiError>
	        <ErrorType>Unauthorized</ErrorType>
	        <ErrorMessage>Unauthorized to scheduled client log
	collection.</ErrorMessage>
	    </ApiError>
	</ApiErrors>
	```

ClientLog API Parameters

Parameter	Type	Description	Possible Values	Notes
id	String	The ID of the user. The ClientLog API uses the id in the name of the log file created on the Finesse server.	—	Maximum of 12 characters. The user must be configured in Unified CCE or Unified CCX.
logData	String	The log data that the client sends to the Finesse server to be stored as a log file.	—	Must not exceed 1,048,576 characters. The user must be authorized to perform the POST operation.

ClientLog API Errors

Status	Error Type	Description
400	Parameter Missing	The logData parameter is not present.
400	Invalid Input	The size of the logData exceeds 1,048,576 characters.
400	Operation Failure	The POST client log operation failed.
401	Authorization Failure	The user is not yet authenticated in the Web Session.
401	Invalid Authorization User Specified	The authenticated user tried to make a request for another user.

Status	Error Type	Description
405	Method Not Allowed	GET or PUT HTTP method not allowed for client-side log collection.

Task Routing APIs

Task Routing APIs provide a standard way to request, queue, route, and handle third-party multichannel tasks in CCE.

Contact Center customers or partners can develop applications using Customer Collaboration Platform and Finesse APIs in order to use Task Routing. The Customer Collaboration Platform Task API enables applications to submit nonvoice task requests to CCE. The Finesse APIs enable agents to sign into different types of media and handle the tasks. Agents sign into and manage their state in each media independently.

Cisco partners can use the sample code available on Cisco DevNet as a guide for building these applications (https://developer.cisco.com/site/task-routing/).

For Finesse, the APIs used for Task Routing include the Media APIs and some of the Dialog and User APIs.

Note This API is only supported for a stand-alone Finesse deployment with Unified CCE and not applicable for coresident Finesse deployment with Unified CCX.

Media

The Media object represents a user's state in a Media Routing Domain (MRD). The Media object is structured as follows:

```
<Media>
    <uri>/finesse/api/User/1001004/Media/5000</uri>
    <description>Chat MRD</description>
    <dialogLogoutAction>CLOSE</dialogLogoutAction>
    <id>5000</id>
    <interruptible>true</interruptible>
    <maxDialogLimit>10</maxDialogLimit>
    <name>Cisco_Chat_MRD</name>
    <ReasonCode>
        <category>NOT_READY</category>
        <code>10</code>
        <forAll>true</forAll/>
        <id>16</id>
        <label>Team Meeting</label>
        <uri>/finesse/api/ReasonCode/16</uri>
    </ReasonCode>
    <reasonCodeId>16</reasonCodeId>
    <routable>true</routable>
    <state>NOT_READY</state>
    <stateChangeTime>2015-09-11T06:55:14.782Z</stateChangeTime>
</Media>
```

Media APIs

Media—Sign In

The Media—Sign In API allows a user to sign in to an individual non-voice Media Routing Domain (MRD) on CCE. If the response is successful, the user is signed in to Finesse and is automatically placed in NOT_READY state and made routable for that MRD. *Routable* means that CCE is allowed to assign an agent tasks in the MRD.

If five consecutive sign-ins fail due to an incorrect password, Finesse blocks access to the user account for a period of 5 minutes.

If a user is already signed in and attempts to sign in again, the user receives an error.

Some parameters used in this API are only known to the Finesse side on which the user signed in. If the user switches sides, the user must sign in again to have this functionality work correctly.

Important Finesse does not support a user staying signed in to both Finesse servers at the same time, through either the REST API or XMPP subscriptions.

The user XMPP presence determines which side a user is signed into, in order to perform actions on the user's behalf. These actions include transferring nonvoice dialogs automatically and either accepting or ignoring interrupts. Finesse transfers nonvoice dialogs automatically if an agent does not accept a dialog within the StartTimeout threshold for the MRD, and if the agent is set to transfer dialogs on sign out in the MRD.

URI:	https://<FQDN>/finesse/api/User/<id>/Media/<mrdId>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Media/5001
Security Constraints:	Users can only act on their own Media objects.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Media>`
	`   <maxDialogLimit>10</maxDialogLimit>`
	`   <state>LOGIN</state>`
	`   <interruptAction>ACCEPT</interruptAction>`
	`   <dialogLogoutAction>CLOSE</dialogLogoutAction>`
	`</Media>`

Request Parameters:	id (required): The ID of the user mrdId (required): The ID of the MRD maxDialogLimit (required): The maximum number of concurrent dialogs this user is allowed to handle in the MRD. Each dialog represents a task. state (required): The new state that the user wants to be in (LOGIN) interruptAction (required): Defines the behavior when an agent is handling a task in an interruptible MRD and is interrupted by a task or call from a non-interruptible MRD. Finesse can ACCEPT the interrupt; the agent is put into INTERRUPTED state and cannot work on dialogs in the interrupted MRD. Finesse can IGNORE the interrupt; the agent's state does not change and the agent can continue to work on the dialogs in the MRD. dialogLogoutAction(optional): Determines whether to TRANSFER or CLOSE active tasks when an agent logs out of the MRD. If not specified, this parameter is set to CLOSE.
Header Parameters:	requestId: A user provided unique string used to correlate originating request with the resulting HTTP response or asynchronous error. This parameter is not part of the resulting event/events.
HTTP Response:	202: Successfully Accepted **Note** The requestId is included in the response header if provided. This response only indicates successful completion of the request. The request is processed and the actual response is sent as part of a media notification. 400: Bad Request (for example, malformed or incomplete request) 400: Parameter Missing 401: Unauthorized (for example, the user is not authenticated in the Web Session) 404: Not Found (for example, the user ID or mrdId is not known) 503: Service Unavailable (for example, the Notification Service is not running)

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorData>1</ErrorData>`
	`        <ErrorMedia>5001</ErrorMedia>`
	`      <ErrorMessage>E_ARM_STAT_AGENT_ALREADY_LOGGED_IN</ErrorMessage>`
	
	`        <ErrorType>Agent already logged into MRD</ErrorType>`
	`    </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Media notification

Asynchronous Errors

If an error occurs after the initial validation is complete, an error notification is sent over XMPP to the Media notification. The requestId is included in the response XML. The ErrorMedia parameter in the ApiError information indicates the Media Routing Domain to which the error applies.

Media—Change State or Sign Out

This API allows a user to change state in or sign out of an individual nonvoice Media Routing Domain.

See Agent States for Nonvoice Media, on page 201 for information about the agent states you can set with this API.

Users can sign out with active tasks. The user's tasks are either automatically transferred or closed, depending on the way the MRD was configured when the user signed in through the Media - Sign In API. To transfer tasks, Finesse resubmits the tasks into the system as new tasks.

URI:	https://<FQDN>/finesse/api/User/<id>/Media/<mrdId>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Media/5001
Security Constraints:	Agents and supervisors can use this API. Users can only act on their own Media objects.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Media>`
	   `<state>LOGOUT</state>`
	`</Media>`

Request Parameters:	id (required): The ID of the user mrdId (required): The ID of the MRD state (required): The new state that the user wants to be in (READY, NOT_READY, LOGIN, or LOGOUT)
Header Parameters:	requestId: A user provided unique string used to correlate originating request with the resulting HTTP response or asynchronous error. This parameter is not part of the resulting event/events.

HTTP Response:	202: Successfully Accepted **Note** The requestId is included in the response header if provided. This response only indicates successful completion of the request. The request is processed and the actual response is sent as part of a media notification. 400: Bad Request (for example, malformed or incomplete request) 401: Unauthorized (for example, the user is not authenticated in the Web Session) 404: Not Found (for example, the user ID or mrdId is not known) 503: Service Unavailable (for example, the Notification Service is not running)

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorData>6</ErrorData>`
	`          <ErrorMedia>5001</ErrorMedia>`
	`         <ErrorMessage>E_ARM_STAT_AGENT_NOT_LOGGED_IN</ErrorMessage>`
	
	`          <ErrorType>Agent is not logged in</ErrorType>`
	`    </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Media notification **Note** The system ignores requests to change agent state from READY to READY; these requests do not trigger a notification.

Asynchronous Errors

If an error occurs after the initial validation is complete, an error notification is sent over XMPP to the Media notification. The requestId is included in the response XML. The ErrorMedia parameter in the ApiError information indicates the Media Routing Domain to which the error applies.

Media—Change Agent State with Reason Code

This API allows a user to change the agent state in an individual non-voice Media Routing Domain, and pass along the code value of a corresponding reason code. Users can use this API only when changing state to NOT_READY or LOGOUT.

URI:	https://<FQDN>/finesse/api/User/<id>/Media/<mrdId>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Media/5001
Security Constraints:	Agents and supervisors can use this API. Users can only act on their own Media objects.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Media>`
	`   <state>NOT_READY</state>`
	`   <reasonCodeId>1001</reaasonCodeId>`
	`</Media>`

Request Parameters:	id (required): The ID of the user mrdId (required): The ID of the Media Routing Domain reasonCodeId (required if reason codes are configured for the given state): The database ID for the reason code state (required): The new state that the user wants to be in (NOT_READY or LOGOUT)
Header Parameters:	requestId: A user provided unique string used to correlate originating request with the resulting HTTP response or asynchronous error. This parameter is not part of the resulting event/events.
HTTP Response:	202: Successfully Accepted **Note** The requestId is included in the response header if provided. This response only indicates successful completion of the request. The request is processed and the actual response is sent as part of a media notification. 400: Bad Request (for example, malformed or incomplete request) 400: Parameter Missing 401: Unauthorized (for example, the user is not authenticated in the Web Session) 404: Not Found (for example, the user ID or mrdId is not known) 503: Service Unavailable (for example, the Notification Service is not running)

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorData>1</ErrorData>`
	`        <ErrorMedia>5001</ErrorMedia>`
	`       <ErrorMessage>E_ARM_STAT_AGENT_ALREADY_LOGGED_IN</ErrorMessage>`
	
	`        <ErrorType>Agent already logged into MRD</ErrorType>`
	`    </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Media notification

Asynchronous Errors

If an error occurs after the initial validation is complete, an error notification is sent over XMPP to the Media notification. The requestId is included in the response XML. The ErrorMedia parameter in the ApiError information indicates the Media Routing Domain to which the error applies.

Media—Change Agent to Routable/Not Routable

The Media—Change Agent to Routable/Not Routable API allows a user to set an agent's routable mode in a Media Routing Domain. Routable mode determines whether CCE can route tasks to an agent in a Media Routing Domain.

When the routable parameter is set to true, the agent is **routable**. CCE can assign task to the agent in that MRD.

When the routable parameter is set to false, the agent is **not routable**. CCE cannot assign tasks to the agent in that MRD.

Make the agent not routable to stop sending tasks to the agent without changing the agent's state to NOT_READY. If an agent changes to NOT_READY state while still working on tasks, those tasks appear ended in CCE reports; time spent working on the tasks after going Not Ready is not counted. You may want to make the agent not routable near the end of the agent's shift, to allow the agent to finish final tasks without being assigned more tasks and to report accurately on those final tasks.

In a RONA situation, in which a task is resubmitted because an agent does not accept a task within the MRD's Start Timeout threshold, Finesse automatically makes the agent not routable.

If a user sets the agent's mode to not routable when an agent has pending incoming tasks or has not started an accepted task, the agent's mode does not change until the agent has started these tasks.

The agent's mode is set to routable automatically when the agent signs in, and when the agent changes to READY state.

URI:	https://<FQDN>/finesse/api/User/<id>/Media/<mrdId>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Media/5001
Security Constraints:	Users can only act on their own Media objects.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	`<Media>` `<routable>true</routable>` `</Media>`
Request Parameters:	id (required): The ID of the user mrdId (required): The ID of the MRD routable(required): Indicates whether CCE can route tasks to the user in the MRD.
Header Parameters:	requestId: A user provided unique string used to correlate originating request with the resulting HTTP response or asynchronous error. This parameter is not part of the resulting event/events.

HTTP Response:	202: Successfully Accepted **Note** The requestId is included in the response header if provided. This response only indicates successful completion of the request. The request is processed and the actual response is sent as part of a media notification. 400: Bad Request (for example, invalid input for parameters) 400: Parameter Missing 401: Unauthorized (for example, the user is not authenticated in the Web Session) 404: Not Found (for example, the user ID or mrdId is not known) 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorData>1</ErrorData>`
	`        <ErrorMedia>5001</ErrorMedia>`
	
	`<ErrorMessage>E_ARM_STAT_ALREADY_IN_REQUESTED_AGENT_MODE</ErrorMessage>`
	
	`        <ErrorType>Agent already in requested mode</ErrorType>`
	`    </ApiError>`
	`</ApiErrors>`

Notifications Triggered:	Media notification

Asynchronous Errors

If an error occurs after the initial validation is complete, an error notification is sent over XMPP to the Media notification. The requestId is included in the response XML. The ErrorMedia parameter in the ApiError information indicates the Media Routing Domain to which the error applies.

Media—Change Agent from Work State to Active

This API allows a user to change the agent state from WORK state to active (READY or NOT_READY), which is automatically computed by Unified CCE. Users can only use this API when an agent state is WORK.

For more information on preventing non-voice task RONAs during CTI reconnect, see *CTI Failover* section in *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

URI:	https://<FQDN>/finesse/api/User/<id>/Media/<mrdId>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Media/5001
Security Constraints:	Agents and supervisors can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Media>`
	`    <mediaConnected>true</mediaConnected>`
	`</Media>`

Request Parameters:	id (required): The ID of the user mrdId (required): The ID of the MRD mediaConnected (required): Indicates media connection to Finesse server. **Note** The `mediaConnected` value can only be set to true and is only intended to be used post initialization of the non-voice channel if the agent is found to be in WORK mode.
Header Parameters:	requestId: User provides a unique string that is used to correlate the originating request with the resulting HTTP response or asynchronous error. This parameter is not part of the resulting event or the events.
HTTP Response:	202: Successfully Accepted **Note** The requestId is included in the response header, if provided. This response only indicates the successful completion of the request. The request is processed and the actual response is sent as part of a media notification. 400: Bad Request (for example, malformed or incomplete request) 401: Unauthorized (for example, the user is not authenticated in the Web Session) 404: Not Found (for example, the user ID or mrdId is not known) 503: Service Unavailable (for example, the Notification Service is not running)

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>finesse.api.media.not_configured</ErrorType>`
	`        <ErrorData>finesse.api.not_found</ErrorData>`
	`      <ErrorMessage>MediaDomain Information Does Not Exists in Finesse`
	` for Id: 500</ErrorMessage>`
	`    </ApiError>`
	`</ApiErrors>`

Asynchronous Errors

If an error occurs after the initial validation is complete, an error notification is sent over XMPP to the Media notification. The requestId is included in the response XML. The ErrorMedia parameter in the ApiError information indicates the Media Routing Domain to which the error applies.

Media—Get Media

This API allows a user to get a copy of a Media object for a specified agent. This API can be used to return only nonvoice Media objects.

URI:	https://<FQDN>/finesse/api/User/<id>/Media/<mrdId>
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Media/5001

Security Constraints:	Users can only act on their own Media objects.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
Request Parameters:	id (required): The ID of the user mrdId (required): The ID of the Media Routing Domain
HTTP Response:	200: Success 400: Bad Request (for example, malformed or incomplete request) 400: Parameter Missing 401: Unauthorized (for example, the user is not authenticated in the Web Session) 404: Not Found (for example, the user ID or mrdId is not known)

Example HTTP Response	**Response if the agent is assigned to skill groups in the Media Routing Domain:**
	`<Media>`
	`    <uri>/finesse/api/User/1001004/Media/5000</uri>`
	`    <description>Chat MRD</description>`
	`    <dialogLogoutAction>CLOSE</dialogLogoutAction>`
	`    <id>5000</id>`
	`    <interruptible>false</interruptible>`
	`    <maxDialogLimit>10</maxDialogLimit>`
	`    <name>Cisco_Chat_MRD</name>`
	`    <ReasonCode>`
	`        <category>NOT_READY</category>`
	`        <code>10</code>`
	`        <forAll>true</forAll/>`
	`        <id>16</id>`
	`        <label>Team Meeting</label>`
	`        <uri>/finesse/api/ReasonCode/16</uri>`
	`    </ReasonCode>`
	`    <reasonCodeId>16</reasonCodeId>`
	`    <routable>true</routable>`
	`    <state>NOT_READY</state>`
	`    <stateChangeTime>2015-09-11T06:55:14.782Z</stateChangeTime>`
	`    <interruptAction>IGNORE</interruptAction>`
	`</Media>`
	Response if the agent is not assigned to skill groups in the Media Routing Domain:
	`<Media>`
	` <uri>/finesse/api/User/1001004/Media/5002</uri>`
	`   <description>Chat MRD</description>`
	`   <id>5002</id>`
	`   <interruptible>false</interruptible>`
	`   <name>Cisco_Chat_MRD2</name>`
	`</Media>`

Example Failure Response:	`<ApiErrors>`
	`  <ApiError>`
	`    <ErrorData>1002001</ErrorData>`
	`    <ErrorMessage>The user specified in the authentication`
	`     credentials and the uri don&apos;t match</ErrorMessage>`
	`     <ErrorType>Invalid Authorization User Specified</ErrorType>`
	`  </ApiError>`
	`</ApiErrors>`

Media—Get List

This API allows a user to get a list of Media objects for all nonvoice Media Routing Domains (MRDs) associated with an agent. The media object also includes the agent's state information for that MRD.

The agent association with an MRD is determined via membership in skill groups associated with the MRD.

URI:	https://<FQDN>/finesse/api/User/<id>/Media
Example URI:	https://finesse1.xyz.com/finesse/api/User/1234/Media
Security Constraints:	Users can only act on their own Media objects.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
Request Parameters:	id (required): The ID of the user
HTTP Response:	200: Success 400: Bad Request (for example, malformed or incomplete request) 400: Parameter Missing 401: Unauthorized (for example, the user is not authenticated in the Web Session) 404: Not Found (for example, the user ID is not known)

Example HTTP Response	`<MediaList>`
	`    <Media>`
	`         ...Full Media Object ...`
	`    </Media>`
	`    <Media>`
	`         ...Full Media Object ...`
	`    </Media>`
	`</MediaList>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`       <ErrorData>1002001</ErrorData>`
	`       <ErrorMessage>The user specified in the authentication`
	`        credentials and the uri don&apos;t match</ErrorMessage>`
	`       <ErrorType>Invalid Authorization User Specified</ErrorType>`
	`   </ApiError>`
	`</ApiErrors>`

MediaDomain—Get List

This API allows a user to get a list of all Media Domain objects configured on Unified CCE.

URI:	https://<FQDN>/finesse/api/MediaDomain
Example URI:	https://finesse1.xyz.com/finesse/api/MediaDomain
Security Constraints:	Only administrators can use this API
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 500: Internal Server Error

Example HTTP Response	`<MediaDomainList>`
	` <MediaDomain>`
	`  <description>Default Media Routing Domain for Cisco_Voice</description>`
	`  <id>1</id>`
	`  <interruptible>false</interruptible>`
	`  <maxDialogDuration>0</maxDialogDuration>`
	`  <name>Cisco_Voice</name>`
	` </MediaDomain>`
	` <MediaDomain>`
	`  <description />`
	`  <id>5000</id>`
	`  <interruptible>true</interruptible>`
	`  <maxDialogDuration>28800</maxDialogDuration>`
	`  <name>Cisco_Chat_MRD</name>`
	` </MediaDomain>`
	` <MediaDomain>`
	`  <description />`
	`  <id>5003</id>`
	`  <interruptible>false</interruptible>`
	`  <maxDialogDuration>60</maxDialogDuration>`
	`  <name>Cisco_Twitter_MRD</name>`
	` </MediaDomain>`
	`</MediaDomainList>`

Example Failure Response:	`<ApiErrors>`
	` <ApiError>`
	`  <ErrorType>Internal Server Error</ErrorType>`
	`  <ErrorMessage>Runtime Exception</ErrorMessage>`
	`  <ErrorData></ErrorData>`
	` </ApiError>`
	`</ApiErrors>`

Agent States for Nonvoice Media

Users can set the following states with the Media APIs:

- LOGIN

- READY
- NOT_READY
- LOGOUT

Users enter the following states automatically while on a task. Users cannot place themselves in these states. For example, agents enter ACTIVE state when they accept a task.

- RESERVED
- ACTIVE
- PAUSED
- INTERRUPTED
- WORK_READY

The agent enters WORK_NOT_READY state automatically if the Finesse server on which the agent is signed in disconnects. When agent signs in again or Finesse side reconnects to CCE, the agent is moved out of the WORK_NOT_READY state. This state cannot be set from the agent desktop.

If an agent is configured to work on a maximum of one task in an MRD, the agent's state in the MRD reflects the agent's activity on that task. However, an agent can be configured to work on several tasks at once in an MRD. The following state hierarchy determines the agent's state in that MRD:

1. LOGIN/LOGOUT
2. READY/NOT_READY
3. INTERRUPTED
4. ACTIVE
5. WORK_READY
6. PAUSED
7. RESERVED

Consider this state hierarchy example. An agent is handling three tasks in an interruptible MRD:

- Task 1 = PAUSED
- Task 2 = WORK_READY
- Task 3 = ACTIVE

Based on the state hierarchy, the agent's overall state in the MRD is ACTIVE. If a task from another MRD then interrupts this MRD, the agent's state in this MRD changes to INTERRUPTED.

The table describes the agent states for nonvoice MRDs.

State	State Information	Allowed Actions
LOGIN	The agent's state immediately after signing in. No tasks are assigned to an agent while in this state. The LOGIN state is a transitive state; LOGIN triggers a change that results in a new state (NOT_READY).	None; the user transitions to NOT_READY automatically
NOT_READY	The agent won't be assigned tasks. The agent enters NOT_READY state automatically after signing in. For accurate task durations in reports, do not change agents to NOT_READY state while they have active tasks. Instead, make the agent not routable to stop assigning tasks to the agent. An agent cannot change to NOT_READY state if the agent has a pending incoming task. The agent has a pending task if Finesse has an offered dialog for that agent.	• READY • LOGOUT
READY	The agent will be assigned tasks. The agent currently doesn't have any tasks. The agent is automatically made routable when the agent enters READY state. When an agent completes all tasks in the MRD, the agent's state returns to the READY.	• NOT_READY • LOGOUT
INTERRUPTED	The agent has been interrupted in this MRD by a task from another MRD. An agent can be interrupted from ACTIVE, WORK_READY, PAUSED, and RESERVED states. The agent cannot perform dialog actions while INTERRUPTED. This state is only applicable for interruptible MRDs in which the agent was configured to accept interrupts when signing into the MRD.	• NOT_READY • LOGOUT
ACTIVE	The agent has accepted at least one offered task. The agent can also have one or more of the following: • Paused tasks • Offered tasks • Tasks for which the agent is performing wrap-up work	• NOT_READY • LOGOUT

State	State Information	Allowed Actions
WORK_READY	The agent is performing wrap-up work for all tasks, or is performing wrap-up work for at least one task and has one or more paused tasks.	• NOT_READY • LOGOUT
PAUSED	The agent has paused all tasks.	• NOT_READY • LOGOUT
RESERVED	The agent has been assigned one or more tasks by CCE, but has not accepted the tasks. The agent does not have active or paused tasks, and is not performing wrap-up work for any tasks.	• NOT_READY • LOGOUT
LOGOUT	The agent signed out of the MRD. If the agent signs out with active tasks, Finesse either closes or transfers the tasks depending on how the dialogLogoutAction parameter was set for the MRD when the agent signed in.	LOGIN
WORK_NOT_READY	The Finesse server on which the agent is signed in disconnected. When an agent fails over to the secondary Finesse server, the agent must sign in to the media again. The agent's state after signing in is determined based on the state of the agent's assigned tasks. If the agent doesn't have tasks, the agent is put in NOT_READY state.	None

Media API Parameters

Note For parameters specified when a user signs in, including maxDialogLimit, interruptAction, and dialogLogoutAction, the setting for the parameter is correct only when the user is signed in.

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the Media object.	—	
description	String	A description of the Media Routing Domain (MRD)	—	Any special XML characters in the description are escaped. For example, "<" is replaced with "&amp;lt;".

Parameter	Type	Description	Possible Values	Notes
id	String	The ID of the user.	—	—
mrdId	String	The ID of the MRD.	—	The size is determined by Unified CCE.
mediaConnected	Boolean	Indicates media connection to Finesse server.	true	Unified CCE only.
interruptible	Boolean	Whether a task in this MRD can be interrupted by a task from another MRD.	true, false	—
maxDialogLimit	Integer	The maximum number of concurrent dialogs this user is allowed to handle in this MRD. Each dialog represents a task.	1–10	The maximum value for this parameter is 10.
name	String	The name of the MRD.	—	—
requestId	String	The earlier sentence was A user provided unique string used to correlate originating request with the resulting HTTP response or asynchronous error. This parameter isn't part of the resulting event/events.	—	—
ReasonCode	Collection	Information about the reason code currently associated with this user.	—	—
-->category	String	The category of the reason code.	NOT_READY	—
-->code	Integer	CTI code associated with this reason code.	—	—
-->forAll	Boolean	Whether the reason code is global (true) or nonglobal (false).	true, false	—
-->id	Integer	The ID of the reason code.	—	—
-->label	String	The label associated with this reason code.	—	—
-->uri	String	The full URI for the reason code.	—	—

Parameter	Type	Description	Possible Values	Notes
reasonCodeId	Integer	The database ID for the reason code that indicates why the user is in the current state in this MRD.	If the user hasn't selected the reason code, this parameter is empty. Otherwise, the value of this parameter is the database ID for the selected reason code.	The value of the reasonCodeId may be -1 in the following cases: • The agent logged out. • No reason codes are configured for the category. • The agent has signed in (transitioned from LOGIN to NOT_READY) • A failover occurred. The agent is in NOT_READY state but Finesse couldn't recover the reasonCode used before failover.
routable	Boolean	Indicates whether CCE can route the tasks to the user in this MRD. When the agent is routable (true), CCE can route tasks to the user. When the agent isn't routable (false), CCE can't route tasks to the agent.	true, false	—
state	String	The state for this user in this MRD.	LOGIN, NOT_READY, READY, LOGOUT, RESERVED, ACTIVE, PAUSED, WORK_READY, INTERRUPTED, WORK_NOT_READY	—

Parameter	Type	Description	Possible Values	Notes
stateChangeTime	String	The time at which the state of the user changed to the current state in this MRD. The format for this parameter is YYYY-MM-DDThh:MM:ss.SSSZ.	—	This parameter is empty if the time of the state change isn't available (if no agent state change notification was received yet).
interruptAction	String	This parameter only applies to interruptible MRDs. It is ignored for noninterruptible MRDs. An agent setting that defines the behavior when an agent is handling a task in an interruptible MRD and is interrupted by a task or call from a non-interruptible MRD. ACCEPT: The MRD accepts the interrupt event. The agent state is INTERRUPTED in the interruptible MRD and the agent can't perform any actions on dialogs in that MRD. IGNORE: The MRD doesn't accept the interrupt event. The agent state doesn't change in the interruptible MRD and the agent can continue to perform actions on dialogs in that MRD.	ACCEPT, IGNORE	This parameter reflects the configured setting only if you're performing a GET on the Finesse server that the user is signed in to.

Parameter	Type	Description	Possible Values	Notes
dialogLogoutAction	String	An agent setting that determines whether active tasks are closed or transferred when an agent logs out of an MRD. CLOSE (default): Active tasks are closed when an agent logs out. Finesse sends Customer Collaboration Platform the task-handled events. CCE determines the correct disposition codes for the closed task. TRANSFER: Active tasks are transferred using Customer Collaboration Platform when an agent logs out. Finesse puts the dialogs in the CLOSED state with the CD_TASK_TRANSFERRED_AGENT_LOGOUT disposition code.	CLOSE, TRANSFER	This parameter reflects the configured setting only if you're performing a GET on the Finesse server that the user is signed in to.

Media API Errors

For synchronous errors, the Media APIs include the requestId in the error response.

Status	Error Type	Description
400	Bad Request	The request is malformed or incomplete.
400	Generic Error	An unaccounted for error occurred. The root cause could not be determined.
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized.
400	Parameter Missing	The state or requestedAction is not provided.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (for example, an agent tries to use an API that only a supervisor or administrator is authorized to use).
401	Invalid Authorization User Specified	The authenticated user tried to make a request for another user.
401	Invalid Supervisor	A supervisor tried to change the state of an agent who does not belong to that supervisor's team.
404	Not Found	The resource specified is invalid or does not exist.

Status	Error Type	Description
503	Service Unavailable	A dependent service is down (for example, the Cisco Finesse Notification Service or Cisco Finesse Database). Finesse is OUT_OF_SERVICE.

Dialog APIs for Nonvoice Tasks

Supported Functionality for Voice and Nonvoice Dialogs

The following are the major differences between supported functionality for voice and nonvoice dialogs:

- Users cannot initiate nonvoice dialogs; nonvoice dialogs are always incoming.
- Nonvoice dialogs can be blind transferred only. Direct transfer is not supported.
- Nonvoice dialogs support only one agent participant. Consult and conference are not supported.

Dialog Object and Parameters for Nonvoice Tasks

The same Dialog object is used for voice calls and nonvoice tasks. The Dialog object includes mediaId and mediaType parameters that indicate the Media Routing Domain with which the dialog is associated.

Some of the Dialog parameters used for voice calls, such as callType and mediaAddressType, are not applicable for nonvoice tasks; these parameters are not returned.

The dialog id format is different for voice calls and nonvoice tasks. The nonvoice dialog id contains underscores (for example, 151635_312_1). Voice dialog ids do not contain underscores (for example, 16804377).

The Dialog section of the Finesse Desktop APIs chapter describes the differences in the Dialog object for voice calls and nonvoice tasks. It also explains the parameters and parameter values used for nonvoice tasks.

Dialog APIs for Nonvoice Tasks

Most Dialog APIs are restricted to voice media.

You can use **Dialog - Take Action on Participant API** to handle nonvoice dialogs. This API supports the following allowable actions for nonvoice tasks.

Action	Description
ACCEPT	Allows an agent to accept an incoming task.
START	Allows an agent to start work on an accepted task.
PAUSE	Allows an agent to pause an active task.
RESUME	Allows an agent to resume a paused task.
TRANSFER	Allows an agent to transfer an accepted, active, or paused task to another Script Selector/dialed number.
WRAP_UP	Allows an agent to perform wrap up work for a task.
CLOSE	Allows an agent to end a task.

Important For nonvoice tasks, dialog actions result only in Finesse reporting the state to CCE. The application is responsible for enforcing that state within the application. For example, if a user pauses an email dialog using the Dialog - Take Action on Participant API, the dialog state PAUSED is reported to CCE. However, if the application still displays the user interface to work on the email, the agent can continue to work on the email. The application must enforce the PAUSED state by preventing agent from working on the email in the user interface.

Notifications

Finesse sends a Dialogs/Media notification when information (or an action) changes for a nonvoice task to which the user belongs.

If a nonvoice dialog operation results in an asynchronous error, the error is returned in a Dialogs/Media notification. The notification includes the error type, error code, and error constant. The ErrorMedia parameter indicates the Media RoutingDomain to which the error applies.

Important For an interruptible Media Routing Domain configured to accept interrupts, Finesse sends only a Media state change when an agent is interrupted in that MRD. It does not send Dialogs/Media notifications with the action list modified to reflect the fact that actions not permitted on the tasks in that media. The state change is the only indication to the Finesse applications that no actions are allowed on the interrupted dialogs.

Interactions with Customer Collaboration Platform

Finesse connects to Customer Collaboration Platform in order to resubmit tasks into the system for these reasons:

- The agent transfers a task.
- A task RONAs while waiting to be accepted by an agent. Finesse automatically resubmits the task to Customer Collaboration Platform.
- An agent signs out with tasks. The agent was configured to transfer tasks on logout. Finesse automatically resubmits the task to Customer Collaboration Platform.

The original dialog is closed with an appropriate disposition code, and the task is resubmitted as a new task request.

For automatic task resubmissions due to RONA and agent logout, the Finesse server on which the agent was last signed in initiates the request.

User APIs for Nonvoice Tasks

Most User APIs are restricted to voice media. Several of them, described here, can be used with nonvoice media.

User- Get List of Dialogs APIs

You can use User - Get List of Dialogs (Nonvoice Only) to get a list of only nonvoice dialogs for a user.

To get a list of both voice and nonvoice dialogs for a user, use the User - Get List of Dialogs (Voice Only by Default) API.

User - Sign Out and User - Change State with Reason Code APIs

You can sign a user out of all Media Routing Domains when the user signs out of the desktop, using either the User - Sign Out API or the User - Change State with Reason Code API.

The desktop sign out fails only if the voice MRD sign out fails; it is not impacted by nonvoice MRD sign out failure.

Single Sign-On

Single Sign-On (SSO) is a mechanism to authenticate users across software systems using a common LDAP identity and this common authentication service provides a token. Multiple applications use this token to authenticate the user across preconfigured applications.

The Single Sign-On (SSO) APIs are used in the Finesse desktop for token related operations and are ready to use in an out of the box Finesse deployment. Third-party desktop applications have to use these APIs independently for SSO token related operations.

Single Sign-On Components

The following are the SSO components:

Identity Provider (IdP)

- IdP is an application that creates, maintains, and manages identity information for users.
- IdP offers the user authentication as a service. Third-party applications (for example, web applications) outsource the user authentication mechanism to a trusted IdP which is configured within the Organization. For example, Active Directory Windows Server.

Cisco Identity Service (IdS)

- Cisco IdS is the common API endpoint for relaying requests to the IdP by generating the authentication token and validating it.
- Cisco IdS implements an authorization endpoint and token endpoint as part of its OAuth (Open Authorization) server implementation.

Token Types

The following are the token types:

- Access Token—It accesses protected resources. Clients are issued an access token that contains identity information for the user that is encrypted by default.

Note For an SSO enabled user, use the access token in the authorization header of the Finesse REST APIs.

Authorization: Bearer <*access token*>

- Refresh Token—It obtains a new access token before the current access token expires. The IdS generates the refresh token.

The refresh and access token are generated as a pair of tokens. When refreshing the access token, the pair of tokens provide an extra layer of security.

You can configure the expiry time of the refresh token and access token in the IdS administration. When the refresh token expires, you cannot refresh the access token.

Cisco Contact Center Components

The following are the Cisco Contact Center components that support SSO:

- Cisco Finesse
- Cisco Unified Intelligence Center

For more information about SSO Solution overview, see https://developer.cisco.com/docs/contact-center-express/#cisco-identity-service-client-sdk-overview.

For more information about the third-party integrations, see https://developer.cisco.com/docs/contact-center-express/#cisco-identity-service-client-sdk-guide/overview.

Single Sign-On APIs

Single Sign-On—Test API

This SSO Test API is used to test the SSO authentication and authorization setup with Finesse.

URI:	https://<FQDN>/desktop/sso/test
Example URI:	https://finesse1.xyz.com/desktop/sso/test
Security Constraints:	Agents and supervisors can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	HTML
HTTP Request:	—
Request Parameters:	—
HTTP Response:	200: Success 400: Bad Request 401: Unauthorized 500: Internal Server Error

Example Response	Response body returned after the SSO test contains an HTML displaying information about the user and token. This HTML also contains a JavaScript that sends the SSO test status, via window postMessage API, to the parent or opener window.
	To get the status of SSO test on an older versions of Internet Explorer or any third-party non-browser clients that do not have this API, use the cookie set as part of HTTP response.
	`COOKIES set as part of response:`
	`ssotest=true`
	`Post message to parent window with below object:`
	`{`
	`    status: "true",`
	`    errorMessage: ""`
	`}`

Example Failure Response:	`COOKIES set as part of response:`
	`ssotest=false`
	`Post message to parent window with below object:`
	`{`
	`    status: "false",`
	`    errorMessage: "AUTH_ERROR"/"NO TOKEN"`
	`}`

Single Sign-On—Fetch Access Token

This API gets the access token and refresh token from the Finesse server.

Note Invoking this API might involve browser redirect to Cisco Identity Service and Identity Provider.

URI:	https://<FQDN>/desktop/sso/token
Example URI:	https://finesse1.xyz.com/desktop/sso/token
Security Constraints:	Agents and supervisors can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	JSON
HTTP Request:	—
Request Parameters:	(Optional) return_user=yes|no (Optional) return_refresh_token=true|false
HTTP Response:	200: Success 400: Bad Request 401: Unauthorized 500: Internal Server Error

Example Response:

Response without any parameter:

```
{"token":"eyJhbGciOiJkaXIiLCJjdHkiOiJKV1QiLCJlbm
    MiOiJBMTI4Q0JDLUhTMjU2In0..lDXjaqAsM89uhdc
    Qt364LA.qXBMK_y58Hkz19k-B8ealJ9LOalB0yNnm9
    vOvKExf8slCpXAPPlJLnNXGD9_-YTGdjs7lPtEcdI—
    hSuDmwxxOhdGZc7ekbAadJ6EItZhOGykCYk_CBF
    mEHKU8-pHV3bdbsUGrCTponA8BMw04-S-N5iuI3v
    u8fuihcNAeRY_9tjl5jvlhHEnD6zrYLDFH8KcO-V2f9
    bcFdxHn3BrZk9tMasrsAJNhm8Uo_kg06PXq9omrTb
    UEKm3f1_lMb3bwqZGXfOO6WLOngsADRTuHren_C
    Tp5gR8r94LpsbXV7gRaEqsCu9kWo3pfxQsu88LNPR
    W6RPcjozupw0A4-jrHBOf_X2XaDquanEbBkZIt9VIJh
    jr6p8bTO5zlH9Z_x7vdMIfEt2pcjqcXKP3NiHlXOaB-tni
    PX_zN8ckGqIKR7L4wBxYmXUj82cnjBNMkcUsbvP9W
    Mb7ihJw0wazl1Tq6WnhtTGeOf0cnorjPm8DOZrcAAjJc
    SDCpudfj5CgE-OwikeSdWURgYTg_k6Kcct71I3olVLT
    c6nFRGcYvclvjCfTc1_ooBQ6ZKI_thq0Apnof235l6drDxG
    sDMPiyop69hWCuMoRRK-KKAXr8xK3fiqKjSse-KMLMG
    rMLZkUsr2Y_Q0YwiEIJk1FJ4n5Qgn-ismhKi-A_Vg3ZicG
    J-YyIcYgcslJGDeqSB10Y0uThqOuMA9eGEHKSlZGLcZ
    BfX5MGv23dEOOxN9_wLkqazF75m5H_23ycLyN0v9d8u
    F7_fe7IWB97cI9nDAhaNBdHBR3XYU5GPSbRRS7GknD
    oWZM_8eTgzc-gFTfYfAJveg_pPr1sSKvWnabqLXUuLDm
    vcVbgA-5UI2Y4HEGKzW85fNOHE9WPpo3cQdxFdRQyH
    fvFCBdTAOiFcIz_uP2nCDB_8oPT7qycm6b58BRJ5EzaTc
    WapskB73w8no1YJadliQ20OYHrDKSs_LJYDeB2iBROS
    UoVocYlW6GwTv0Ko7NsLv3OtGc_I.Fre8fhy_Y4u11tIfNo6
    fIA","expires_in":300}
```

Response with **`return_user=yes`** parameter:

```
{"token":"eyJhbGciOiJkaXIiLCJjdHkiOiJKV1QiLCJlbm
    MiOiJBMTI4Q0JDLUhTMjU2In0..lDXjaqAsM89uhdcQ
    t364LA.qXBMK_y58Hkz19k-B8ealJ9LOalB0yNnm9v
    OvKExf8slCpXAPPlJLnNXGD9_-YTGdjs7lPtEcdI—
    hSuDmwxxOhdGZc7ekbAadJ6EItZhOGykCYk_CBFm
    EHKU8-pHV3bdbsUGrCTponA8BMw04-S-N5iuI3vu8
    fuihcNAeRY_9tjl5jvlhHEnD6zrYLDFH8KcO-V2f9bc
    FdxHn3BrZk9tMasrsAJNhm8Uo_kg06PXq9omrTbUE
    Km3f1_lMb3bwqZGXfOO6WLOngsADRTuHren_CTp
    5gR8r94LpsbXV7gRaEqsCu9kWo3pfxQsu88LNPR
    W6RPcjozupw0A4-jrHBOf_X2XaDquanEbBkZIt9V
    IJhjr6p8bTO5zlH9Z_x7vdMIfEt2pcjqcXKP3NiHlXOaB-
    tniPX_zN8ckGqIKR7L4wBxYmXUj82cnjBNMkcUsbvP
    9WMb7ihJw0wazl1Tq6WnhtTGeOf0cnorjPm8DOZrcA
    AjJcSDCpudfj5CgE-OwikeSdWURgYTg_k6Kcct71I3ol
    VLTc6nFRGcYvclvjCfTc1_ooBQ6ZKI_thq0Apnof235l6
    drDxGsDMPiyop69hWCuMoRRK-KKAXr8xK3fiqKjSse-
    KMLMGrMLZkUsr2Y_Q0YwiEIJk1FJ4n5Qgn-ismhKi-A
    _Vg3ZicGJ-YyIcYgcslJGDeqSB10Y0uThqOuMA9e
    GEHKSlZGLcZBfX5MGv23dEOOxN9_wLkqazF75m5H
    _23ycLyN0v9d8uF7_fe7IWB97cI9nDAhaNBdHBR3XYU
    5GPSbRRS7GknDoWZM_8eTgzc-gFTfYfAJveg_pPr1s
    SKvWnabqLXUuLDmvcVbgA-5UI2Y4HEGKzW85fNO
    HE9WPpo3cQdxFdRQyHfvFCBdTAOiFcIz_uP2nCDB_8
    oPT7qycm6b58BRJ5EzaTcWapskB73w8no1YJadliQ20O
    YHrDKSs_LJYDeB2iBROSUoVocYlW6GwTv0Ko7NsLv3
    OtGc_I.Fre8fhy_Y4u11tIfNo6fIA","expires_in":49,
    "user_id":"1001001",
    "realm":"finesse.com",
    "user_principal":"1001001@finesse.com"}
```

	Response with **`return_refresh_token=true`** parameter:
	`{"token":"eyJhbGciOiJkaXIiLCJjdHkiOiJKV1QiLCJlbm`
	`MiOiJBMTI4Q0JDLUhTMjU2In0..lDXjaqAsM89uhdcQ`
	`t364LA.qXBMK_y58Hkz19k-B8ealJ9LOalB0yNnm9v`
	`OvKExf8slCpXAPPlJLnNXGD9_-YTGdjs7lPtEcdI—`
	`hSuDmwxxOhdGZc7ekbAadJ6EItZhOGykCYk_CBFm`
	`EHKU8-pHV3bdbsUGrCTponA8BMw04-S-N5iuI3vu8`
	`fuihcNAeRY_9tjl5jvlhHEnD6zrYLDFH8KcO-V2f9bc`
	`FdxHn3BrZk9tMasrsAJNhm8Uo_kg06PXq9omrTbUE`
	`Km3f1_lMb3bwqZGXfOO6WLOngsADRTuHren_CTp`
	`5gR8r94LpsbXV7gRaEqsCu9kWo3pfxQsu88LNPR`
	`W6RPcjozupw0A4-jrHBOf_X2XaDquanEbBkZIt9V`
	`IJhjr6p8bTO5zlH9Z_x7vdMIfEt2pcjqcXKP3NiHlXOaB-`
	`tniPX_zN8ckGqIKR7L4wBxYmXUj82cnjBNMkcUsbvP`
	`9WMb7ihJw0wazl1Tq6WnhtTGeOf0cnorjPm8DOZrcA`
	`AjJcSDCpudfj5CgE-OwikeSdWURgYTg_k6Kcct71I3ol`
	`VLTc6nFRGcYvclvjCfTc1_ooBQ6ZKI_thq0Apnof235l6`
	`drDxGsDMPiyop69hWCuMoRRK-KKAXr8xK3fiqKjSse-`
	`KMLMGrMLZkUsr2Y_Q0YwiEIJk1FJ4n5Qgn-ismhKi-A`
	`_Vg3ZicGJ-YyIcYgcslJGDeqSB10Y0uThqOuMA9e`
	`GEHKSlZGLcZBfX5MGv23dEOOxN9_wLkqazF75m5H`
	`_23ycLyN0v9d8uF7_fe7IWB97cI9nDAhaNBdHBR3XYU`
	`5GPSbRRS7GknDoWZM_8eTgzc-gFTfYfAJveg_pPr1s`
	`SKvWnabqLXUuLDmvcVbgA-5UI2Y4HEGKzW85fNO`
	`HE9WPpo3cQdxFdRQyHfvFCBdTAOiFcIz_uP2nCDB_8`
	`oPT7qycm6b58BRJ5EzaTcWapskB73w8no1YJadliQ20O`
	`YHrDKSs_LJYDeB2iBROSUoVocYlW6GwTv0Ko7NsLv3`
	`OtGc_I.Fre8fhy_Y4u11tIfNo6fIA","refresh_token":"`
	`eyJhbGciOiJkaXIiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMTI4Q0J`
	`DLUhTMjU2In0..gdULcCw-3nh_R-FnIHPXDQ.0kYaOssJH76ro`
	`2iD5JCYBuZqVMFrDpA0FDeK_bm9ClcuJaaU4vNWnww7r6G7qT4A`
	`FdQZqnIMq0kBSzpbfKVIchOs95usysQA3IYi2d8dlEblIkdRiPW`
	`YWhNbhj_KjcNM-Rjt9SbinW3dfl8NL_OB0MTs0HYa8DwdcXnP_`
	`62QonodfHAiGVH9i8VgoHKBjCLUTNcUbVX43pdxy`
	`OvLwuOOZh4Uzp-8L1dn16PE0wU7wBvyl2J6xTrAEZ1Beya2pCP-zn8e`
	`CYTqAEiuzbm-DVP9b4G7w4qusQ_Z347b0Oa782o7_Ui4_mpbt2kaOP5w`
	`YByR0ftCkGLsAnPyJ5BpB89Sd2TnXmRWMhc72vqu1AM67UimxUpjWxVDZ`
	`em0QowbYKLdEWZSMUP744hLgrGFEKdanWQKFKoOMzqhprfHCz3VZYs6kZv`
	`hvWRsmoU8hI9j0qklfNuhJkCs-UeF6GnlN7FNsxEfHAvcXgF_oxGFQcEjoe`
	`k0cYPzUWhIqV94bapeYm6sRH7d38QGBRm2D-mkdsQyMcQvH66NJ3uYPm1Inw`
	`NoNQuaS0feo4OTpms5CIt49jblNrl7k5Kpk6Q7DMKcCuyKK4OyxPY9uN-7y2l`
	`3XnmtG3jhqOTjQ9t_1YdBh-RFHbv6M2KWWIHZ57CpRpSNCo-wN1VRaOkbQaLw`
	`LwEU_Hk-Cp4.RRHsyddqhFEtH0xvRc5EjA","expires_in":49}`

Example Failure Response:	`{"error":"invalid_redirectUri","error_description":"Invalid Redirect`
	
	`URI."}`

Note When you use the **`return_refresh_token=true`** query parameter in this API, access token and refresh token cookies are not added to the response. All information is provided as part of the response body, which can be directly used by the third-party clients.

Use this query parameter when third-party clients use Cisco Finesse SSO APIs alongside Finesse desktop in the same browser. Using this query parameter prevents agent logout from Finesse desktop due to desktop cookie overriding by third-party clients.

Single Sign-On—Refresh Existing Access Token

This API allows a user to refresh an existing access token that is about to expire.

Note

- Third-party applications have to refresh the access token after 75% of the token expiry time is elapsed.
- Invoking this API might involve browser redirect to Cisco Identity Service and Identity Provider.

URI:	https://<FQDN>/desktop/sso/token
Example URI:	https://finesse1.xyz.com/desktop/sso/token
Security Constraints:	Agents and supervisors can use this API.
HTTP Method:	POST
Content Type:	application/x-www-form-urlencoded
Input/Output Format:	Application/JSON
HTTP Request:	—

Request URI Parameters:	token=<token value>
	refresh-token=<refresh token value>
	(Optional) return_user=yes

HTTP Response:	200: Success 400: Bad Request 401: Unauthorized 500: Internal Server Error

Example Response	`{"token": "eyJhbGciOiJkaXIiLCJjdHkiOiJKV1QiLCJlbmMiOiJBM`
	`    TI4Q0JDLUhTMjU2In0..521UM8q8d7wM5naKgWzPhA.NkhEH`
	`    7SatpXPOVqQobJstaZ51HBcMTcIej5qdIJ0ZwjCnV7u8iKGcv7t`
	`    5cLYruV6WZFJn8z7iSckXdduDqmRserhBDnbpk-gd5jqNj9r2ZS`
	`    tfeBZIx6Phng6EMWUjtK9cbrO79MenQ7u7Y3Hhe7P7qvQiaTw`
	`    keUw7No09NFGat-ICzhHbTF8D4WKFhFefw1J-q55ktcdD-CmM`
	`    s-KXYrmA8DLltjF9ii9dCYHFfC2nKBETzdYWR2ple4B6_Lv0np`
	`    g8OSU53LyTT3ObHm6TvWZ09KYrWUWMKNFas73Gx7rYro4`
	`    C7Tc4pYb9ZfJmkcT6coRIocMteYCrqCy7ufRqO-BPObNIah_J`
	`    o2VQ_wwo-5wE-cMUUDpGa5X2nMtP2YUH4sb7b_SHX9Xq_w6`
	`    cwLRcBiDXjyGl7Smk1RzF1aXj2A9R06a71VjzmUsjq4UtrT7_IfY`
	`    s9RrFX9jhnXX1VB8Dqgh-Pnb16rsskRg7TPP4EV9fwDSbhA-`
	`    oMrMKqFz5BFWMhNaFCHtJQWtXxNRK802ybyzXwR3KGeINS`
	`    D3dOGj2vWRpnhuTB9veHr9InSrc2s67rspguN7YX2bkIEEQNBC`
	`    Y3X5rf_UMyGSlPvlArh6b-_yZXk62kXmYJWJ7g1uTRwTaou87C`
	`    j83fqdaIOYMNIOeZhZqDmKDOZqMmVW_Aj-9-Tn0lTXkKmsPvqt`
	`    oJYCN1T_3fZrvhzJLImy0whXgEtxc88MYNOCsuPSkIuCRNpoO`
	`    GgWXATdF1GHPUnQPStW2GsZEfbdY5R1X9x3SZXtngh4XFM`
	`    gYtMjP129X8pvAT_AY35JtRzpdryRPdAYrEc72tkY_xWLBahpS`
	`    AKrcX7x8gtMRZmV5HlKs7_sW1amje0gaMKFlqh8i56XWbwnsU`
	`    SdKLC-LZDtvWZ5wYuHPY1CSwC0oT9lHytWBXo3GSXSv`
	`    1iqy75ud6KrvrJg3WG2k_2biqxpc0S9MsATT2WGtGBt5ko2wEcn6`
	`    A.l_JfM6gAelSswEeGFAOKwg",`
	`    "expires_in": 300}`

Example Failure Response:	`{"errorType": "AUTH_ERROR",`
	`    "errorData": "refresh-token",`
	`    "errorMessage": "Invalid Token"}`

Note If the token was initially fetched with the **`return_refresh_token=true`** query parameter, the refresh token query parameter with the value of the current refresh token is mandatory.

Troubleshooting Steps

1. Validate that the requests are reaching Finesse by checking the Finesse tomcat_access_logs.
2. For any errors on the token refresh, check the Finesse Valve logs.
3. If the request reached Finesse access logs and there are no errors in Finesse Valve logs, check the IdS error logs for more details. It contains the reason for the error.

Single Sign-On—Get User Authentication Mode

This API allows a client to get the authentication mode of a user in a Unified CCE deployment that is in hybrid mode (SSO and non-SSO). This API uses either the username or userId and it does not require authentication.

Note This API does not require HTTP authentication. The third-party integrations must configure this API to determine which authentication mode (SSO or non-SSO) is configured for the user. If required, this API can be disabled using CLI. By default, the CLI sets the value of this property as *true*.

utils finesse set_property webservices enableUserAuthMode {*true*|*false*}

For more information, see *Service Properties* section in *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

URI:	https://<FQDN>/finesse/api/UserAuthMode/<username> https://<FQDN>/finesse/api/UserAuthMode/<userId>
Example URI:	https://finesse1.xyz.com/finesse/api//UserAuthMode/myName https://finesse1.xyz.com/finesse/api//UserAuthMode/1234
Security Constraints:	All users can use this API without authentication.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
Request Parameters:	—
HTTP Response:	200: Success 403: Forbidden

Example Response	`<UserAuthMode>`
	`    <authMode>NON_SSO</authMode>`
	`</UserAuthMode>`

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>Forbidden</ErrorType>`
	`        <ErrorMessage>UserAuthModeService is disabled</ErrorMessage>`
	`    <ApiError>`
	`</ApiErrors>`

Single Sign-On Parameters

Parameter	Type	Description	Possible Values	Notes
username	String	The username that is configured at ADFS.	—	—
userId	String	The userId is the peripheralId that is configured in Unified CCE.	—	—
authMode	String	Information about the user authentication mode.	SSO or non-SSO	—

Single Sign-On API Errors

Status	Error Type	Description
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.

Client Integration

Clients can use the Finesse REST APIs in SSO mode. For thick client integrations, the following are browser like behaviors that thick clients must ensure to exhibit:

- Follow server issued redirects.
- Store and forward cookies.
- Honor the various cookie attributes.
- Run JavaScript in HTML responses.

Procedure

Step 1 Use the API to get the system's authentication mode. The authentication mode can be found in the response as the value of the systemAuthMode.

Note If the system's authentication mode is SSO, then you can skip step 2.

Step 2 Use the **Single Sign-On—Get User Authentication Mode** to get a specific user's authentication mode.

Note You must use browser components that allow redirections and IdP form submissions. You cannot use Postman or AJAX for the Single Sign-On APIs.

Step 3 Use the **Single Sign-On—Get User Authentication Mode** with the `return_user` query parameter set to **`yes`** to get the user's access token.

The username must be provided in a cookie or a URL query parameter with a key of cc_username. The value is a URL encoded username, which can be the loginName or peripheralId for whom the token is requested.

Example:

```
https://finesse1.xyz.com/desktop/sso/token?return_refresh_token=true
```

The result of the API request will redirect the request to the IdS page which then redirects to the IdP page.

Step 4 On the IdP page, enter the **username** and the **password**.

- On successful authentication, the response body contains the access token and user_id value. The access token returns the token in the JWT format.

  Example Response:

```
{"token":"eyJhbGciOiJkaXIiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..0
KelINtPSTQJGHthEo6GGg.iJl7anAh8Edt07pOQmHZ7BLgK-ozMiy5Fy42Pkj8FQ3xUMQvq5coGwSnHCEr1
deuNFt5i6685L5aHZzGe3VChWRPOnHveKaOEbgjgdNAhFXnIF033H4hZ-sKf zGSdIiXDIqv7llOGhzwDWw
jYA7icEgIqaTttu5VVMPsI5eCwwd8uf1SXA7_rNU9bM1Kkra-v PRyqJX4h2gR-1vbk1q3L1GEtaAnWqDIe
4MAZELtuD-J5O4Kid26lwKzdq5PL_PH51-lyw_Mz ds0JiE4ZWcG_JmAa4BtZfKGs1z4S6Laj7scxi7WW7u
5-lfBsn6ixhouC3Jdz4N3FJQZ6IizQ 0tUwOWb0HoD6tsU6mH4r5tYeRjtGliTJff71BIAkZl6N1fCB0ZN2
P5lPMWPqfIgqw8r0H5Ar 1xDNpmfs_b0e0lHwfCczWKqn96wFIUeP0IBpk2uv7-8C5NvM6U72Vbs9SjUH7T
3b8zZzt9mg Hnu2fdluW5OfrdLxF6BGiI20_p6jjI0D7HPrqpX-I7RGSWB79fEFm0IOFAEy04kwvRJBJx8hI
Mu-2AHc38NW7i-mZVzbE28K2pPwyeFMR6UWKIl0ztDsAyEQA89DI1bOukFa57CDQzF4mR5szmczaLeoXTAC
hY8qPMvAEjMtAvSIQSwb9W-D6HoxBQM6gF-Sth1eD2n82gkbb1-Gg4GF3gzoSa-_kf1sYj62mF4PUsJVN4_
cFXRgzdyhjkDielwabBJPbt0oAXR2qj3qH7TBxLj4hYdKbPq2bDd9eFNeMhTMX49hYrONGWaSz3CdBb3fby
177ALj-AHlGU2mbZ1Ofa3hUj5h1H3p7QwGWGs1Ka56DHK3cTcPszAvMdtVF0MZsK9ODr0gJVFFKPvoY2alb
d3xG6rbbQmvWGoSYWgT7l2dzUYokEOEjN6halaZmOBcnjxS-NeqCRrve-22zwqFwD-fEdjRf9ATtq32UcB_
RmlnubDJOndJQ.5kix_gQZGr6xGye3cUE18g","refresh_token":"eyJhbGciOiJkaXIiLCJjdHkiOiJK
V1QiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..gdULcCw-3nh_R-FnIHPXDQ.0kYaOssJH76ro2iD5JCYBuZ
qVMFrDpA0FDeK_bm9ClcuJaaU4vNWnww7r6G7qT4AFdQZqnIMq0kBSzpbfKVIchOs95usysQA3IYi2d8dlE
blIkdRiPWYWhNbhj_KjcNM-Rjt9SbinW3dfl8NL_OB0MTs0HYa8DwdcXnP_62QonodfHAiGVH9i8VgoHKBj
CLUTNcUbVX43pdxyOvLwuOOZh4Uzp-8L1dn16PE0wU7wBvyl2J6xTrAEZ1Beya2pCP-zn8eCYTqAEiuzbm-
DVP9b4G7w4qusQ_Z347b0Oa782o7_Ui4_mpbt2kaOP5wYByR0ftCkGLsAnPyJ5BpB89Sd2TnXmRWMhc72vq
u1AM67UimxUpjWxVDZem0QowbYKLdEWZSMUP744hLgrGFEKdanWQKFKoOMzqhprfHCz3VZYs6kZvhvWRsmo
U8hI9j0qklfNuhJkCs-UeF6GnlN7FNsxEfHAvcXgF_oxGFQcEjoek0cYPzUWhIqV94bapeYm6sRH7d38QGB
Rm2D-mkdsQyMcQvH66NJ3uYPm1InwNoNQuaS0feo4OTpms5CIt49jblNrl7k5Kpk6Q7DMKcCuyKK4OyxPY9
uN-7y2l3XnmtG3jhqOTjQ9t_1YdBh-RFHbv6M2KWWIHZ57CpRpSNCo-wN1VRaOkbQaLwLwEU_Hk-Cp4.RRH
syddqhFEtH0xvRc5EjA","expires_in":3564,"user_id":"sjefferson","realm":"finesse.com",
"user_principal":"sjefferson@finesse.com"}
```

- The response also contains the token expiry time in seconds.

Note In a Unified CCX deployment, the username, loginName, and loginId can be used interchangeably for the Finesse REST API calls.

Step 5 This step is for Unified CCE deployments only. In a Unified CCE deployment, the user_id obtained from the IdS token can be either the loginName or the peripheralId (loginId). The Finesse REST APIs can only accept the loginId. Use the **User—Get User Id from loginName** API to get the loginId from the user_id of the IdS token.

Example:

```
https://finesse1.xyz.com/finesse/api/User/sjefferson
```

Example Response:

```
<User>
    <dialogs>/finesse/api/User/1001002/Dialogs</dialogs>
    <extension></extension>
    <firstName>AGENT</firstName>
    <lastName>98411</lastName>
    <loginId>98411</loginId>
    <loginName>sjefferson</loginName>
    <mediaType>1</mediaType>
    <pendingState></pendingState>
    <reasonCodeId>-1</reasonCodeId>
    <roles>
        <role>Agent</role>
    </roles>
    <settings>
        <wrapUpOnIncoming>OPTIONAL</wrapUpOnIncoming>
    </settings>
    <state>LOGOUT</state>
    <stateChangeTime></stateChangeTime>
    <teamId>5000</teamId>
    <teamName>FunctionalAgents</teamName>
    <uri>/finesse/api/User/sjefferson</uri>
    <wrapUpTimer>30</wrapUpTimer>
</User>
```

Step 6 Get the loginId from the **loginId** field.

All subsequent Finesse REST API requests must use the loginId from the <User> response, instead of the username/user_id/loginName.

Example:

To login the agent *sjefferson* using the Finesse REST API, you must use the loginId of 98411.

```
https://finesse1.xyz.com/finesse/api/User/98411
```

```
<User>
    <state>LOGIN</state>
    <extension>98411</extension>
</User>
```

Step 7 To avoid the authentication and authorization flow again, the access token must be refreshed before the expiry time. Use the Single Sign-On—Refresh Existing Access Token with the `return_user` query parameter set to **`yes`** to refresh the user's access token.

The username must be provided in a cookie or a URL query parameter with a key of cc_username. The value is a URL encoded username, which can be the loginName or peripheralId for whom the token is requested.

Example Response:

```
{"token":"eyJhbGciOiJkaXIiLCJjdHkiOiJKV1QiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..0
KelINtPSTQJGHthEo6GGg.iJl7anAh8Edt07pOQmHZ7BLgK-ozMiy5Fy42Pkj8FQ3xUMQvq5coGwSnHCEr1
deuNFt5i6685L5aHZzGe3VChWRPOnHveKaOEbgjgdNAhFXnIF033H4hZ-sKf zGSdIiXDIqv7llOGhzwDWw
jYA7icEgIqaTttu5VVMPsI5eCwwd8uf1SXA7_rNU9bM1Kkra-v PRyqJX4h2gR-1vbk1q3L1GEtaAnWqDIe
4MAZELtuD-J5O4Kid26lwKzdq5PL_PH51-lyw_Mz ds0JiE4ZWcG_JmAa4BtZfKGs1z4S6Laj7scxi7WW7u
5-lfBsn6ixhouC3Jdz4N3FJQZ6IizQ 0tUwOWb0HoD6tsU6mH4r5tYeRjtGliTJff71BIAkZl6N1fCB0ZN2
P5lPMWPqfIgqw8r0H5Ar 1xDNpmfs_b0e0lHwfCczWKqn96wFIUeP0IBpk2uv7-8C5NvM6U72Vbs9SjUH7T 3b8zZzt9mg
Hnu2fdluW5OfrdLxF6BGiI20_p6jjI0D7HPrqpX-I7RGSWB79fEFm0IOFAEy04kwvRJBJx8hI
Mu-2AHc38NW7i-mZVzbE28K2pPwyeFMR6UWKIl0ztDsAyEQA89DI1bOukFa57CDQzF4mR5s zmczaLeoXTAC
hY8qPMvAEjMtAvSIQSwb9W-D6HoxBQM6gF-Sth1eD2n82gkbb1-Gg4GF3gzo Sa-_kf1sYj62mF4PUsJVN4_
cFXRgzdyhjkDielwabBJPbt0oAXR2qj3qH7TBxLj4hYdKbPq2b Dd9eFNeMhTMX49hYrONGWaSz3CdBb3fby1
77ALj-AHlGU2mbZ1Ofa3hUj5h1H3p7QwGWGs1Ka 56DHK3cTcPszAvMdtVF0MZsK9ODr0gJVFFKPvoY2albd3
xG6rbbQmvWGoSYWgT7l2dzUYokEOE jN6halaZmOBcnjxS-NeqCRrve-22zwqFwD-fEdjRf9ATtq32UcB_Rml
nubDJOndJQ.5kix_gQZ Gr6xGye3cUE18g","expires_in":3564,"user_id":"sjefferson",
"realm":"finesse.com","user_principal":"sjefferson@finesse.com"}
```

TeamMessage

The TeamMessage object represents messages that can be sent by the supervisor or the Finesse administrator to any or all teams. It contains the URI, team message, and id of the sender. The supervisor or administrator uses the TeamMessage APIs to create or delete a team message, return all active messages for a team, and return all messages created by a user.

The TeamMessage object is structured as follows:

```
<TeamMessage>
    <uri>/finesse/api/TeamMessage/be1598bb-bb2a-4dfc-8c01-91ec10b029af</uri>
    <id>be1598bb-bb2a-4dfc-8c01-91ec10b029af</id>
    <createdBy>
        <id>1001050</id>
        <firstName>AGENT</firstName>
        <lastName>1001050</lastName>
    </createdBy>
    <createdAt>1537418173</createdAt>
    <duration>100</duration>
    <content>content 4</content>
    <teams>
        <team>5052</team>
        <team>5000</team>
    </teams>
</TeamMessage>
```

TeamMessage APIs

TeamMessage—Get Team Message

This API allows the user to get a copy of a TeamMessage object.

URI:	https://<FQDN>/finesse/api/TeamMessage/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/TeamMessage/123
Security Constraints:	Supervisors or administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 404: Not Found 500: Internal Server Error

Example Response:	`<TeamMessage>`
	
	`<uri>/finesse/api/TeamMessage/be1598bb-bb2a-4dfc-8c01-91ec10b029af</uri>`
	
	`    <id>be1598bb-bb2a-4dfc-8c01-91ec10b029af</id>`
	`    <createdBy>`
	`        <id>1001050</id>`
	`        <firstName>AGENT</firstName>`
	`        <lastName>1001050</lastName>`
	`    </createdBy>`
	`    <createdAt>1537418173</createdAt>`
	`    <duration>100</duration>`
	`    <content>content 4</content>`
	`    <teams>`
	`        <team>5052</team>`
	`        <team>5000</team>`
	`    </teams>`
	`</TeamMessage>`

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>Not Found</ErrorType>`
	`        <ErrorData>finesse.api.not_found</ErrorData>`
	`        <ErrorMessage>Message with ID`
	`06f381e6-10ee-47a9-9b36-1c2d7b62db08 not found.</ErrorMessage>`
	`    </ApiError>`
	`</ApiErrors>`

TeamMessage—Get List

This API allows the user to get a list of all Team Messages that are created by the user.

URI:	https://<FQDN>/finesse/api/TeamMessages?createdBy=<id>
Example URI:	https://finesse1.xyz.com/finesse/api/TeamMessages?createdBy=1001050
Security Constraints:	Administrator and supervisor who created the message can use this API. For Administrators, if the createdBy parameter has no value, it returns all active messages.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 404: Not Found 500: Internal Server Error

<table>
<tr><td><b>Example Response:</b></td><td>

```
<TeamMessages>
    <TeamMessage>

<uri>/finesse/api/TeamMessage/be1598bb-bb2a-4dfc-8c01-91ec10b029af</uri>

        <id>be1598bb-bb2a-4dfc-8c01-91ec10b029af</id>
        <createdBy>
            <id>1001050</id>
            <firstName>AGENT</firstName>
            <lastName>1001050</lastName>
        </createdBy>
        <createdAt>1537418173</createdAt>
        <duration>100</duration>
        <content>content 4</content>
        <teams>
            <team>5052</team>
            <team>5000</team>
        </teams>
    </TeamMessage>
    <TeamMessage>

<uri>/finesse/api/TeamMessage/c652fb4f-1f1a-48c8-bc77-2cbab3c9d231</uri>

        <id>c652fb4f-1f1a-48c8-bc77-2cbab3c9d231</id>
        <createdBy>
            <id>1001050</id>
            <firstName>AGENT</firstName>
            <lastName>1001050</lastName>
        </createdBy>
        <createdAt>1537418172</createdAt>
        <duration>100</duration>
        <content>content 4</content>
        <teams>
            <team>5052</team>
            <team>5000</team>
        </teams>
    </TeamMessage>
    <TeamMessage>

<uri>/finesse/api/TeamMessage/ea74a0db-efcf-4651-84b1-1d2119509e9f</uri>

        <id>ea74a0db-efcf-4651-84b1-1d2119509e9f</id>
        <createdBy>
            <id>1001050</id>
            <firstName>AGENT</firstName>
            <lastName>1001050</lastName>
        </createdBy>
        <createdAt>1537418177</createdAt>
        <duration>100</duration>
        <content>some content 4</content>
        <teams>
            <team>5052</team>
            <team>5000</team>
        </teams>
    </TeamMessage>
</TeamMessages>
```

</td></tr>
<tr><td><b>Example Failure Response:</b></td><td>

```
<ApiErrors>
    <ApiError>
        <ErrorType>Unauthorized</ErrorType>
        <ErrorMessage>Not authorized to access this
resource.</ErrorMessage>
    </ApiError>
</ApiErrors>
```

</td></tr>
</table>

TeamMessage—Create a Team Message

This API allows the user to create a TeamMessage object.

URI:	https://<FQDN>/finesse/api/TeamMessage
Example URI:	https://finesse1.xyz.com/finesse/api/TeamMessages
Security Constraints:	Supervisors or administrators can use this API.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 207: Partially succeeded **Note** 207 indicates that one of the operations (Create or Delete) has succeeded but publishing to the alternate node might have failed due to DB replication issues. In this case, the message broadcasted by a supervisor (logged in to one of the Finesse nodes) might not be displayed to the agents logged in to the alternate Finesse node. 401: Authorization Failure 404: Not Found 500: Internal Server Error 503: Service Unavailable

Example Response:	`<TeamMessage>`
	`    <duration>100</duration>`
	`    <content>content 3</content>`
	`    <teams>`
	`        <team>5000</team>`
	`        <team>5052</team>`
	`    </teams>`
	`</TeamMessage>`

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>System Resource Limit Exceeded</ErrorType>`
	`        <ErrorData>teammessage.max.limit.exceeded</ErrorData>`
	`       <ErrorMessage>MAX_ACTIVE_MESSAGE_LIMIT_EXCEEDED</ErrorMessage>`
	
	`    </ApiError>`
	`</ApiErrors>`

TeamMessage—Delete a Team Message

This API allows the supervisor who created the Team Message or administrator, to delete a Team Message. The supervisor or administrator can reference the existing TeamMessage object by its ID.

URI:	https://<FQDN>/finesse/api/TeamMessage/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/TeamMessage/be1598bb-bb2a-4dfc-8c01-91ec10b029af
Security Constraints:	Supervisor who created the Team Message or the administrator can use this API.
HTTP Method:	DELETE
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 207: Partially succeeded **Note** 207 indicates that one of the operations (Create or Delete) has succeeded but publishing to the alternate node might have failed due to DB replication issues. In this case, the message broadcasted by a supervisor (logged in to one of the Finesse nodes) might not be displayed to the agents logged in to the alternate Finesse node. 401: Authorization Failure 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>Not Found</ErrorType>`
	`        <ErrorData>finesse.api.not_found</ErrorData>`
	`        <ErrorMessage>Message with ID`
	`06f381e6-10ee-47a9-9b36-1c2d7b62db08 not found.</ErrorMessage>`
	`    </ApiError>`
	`</ApiErrors>`

TeamMessage API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the TeamMessage object.	—	
id	String	The unique identifier for the TeamMessage.	—	
createdBy	String	The Agent ID of the creator of the TeamMessage.	—	
createdAt	Integer	The UTC time of the TeamMessage posted in seconds.	—	

Parameter	Type	Description	Possible Values	Notes
duration	Integer	The time the TeamMessage is displayed in seconds.	—	
content	String	The content of the TeamMessage.	—	A maximum of 255 characters are supported.
team	Integer	The ID of the particular team.	—	

TeamMessage API Errors

Status	Error Type	Description
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session).
404	Not Found	The resource specified is invalid or does not exist.
404	User Not Found	The user ID provided is invalid or is not recongnized. No such user exists in CTI.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

CHAPTER 4

Cisco Finesse Configuration APIs

SystemConfig

The SystemConfig object is a container element that holds the Finesse system configuration, including details about the primary and backup CTI servers.

Note SystemConfig APIs apply only to Finesse deployments with Unified CCE. Because you need not configure these settings for Finesse with Unified CCX, these APIs are not supported for deployments with Unified CCX.

The SystemConfig object is structured as follows:

```
<SystemConfig>
   <uri>/finesse/api/SystemConfig</uri>
   <cti>
      <host></host>
      <port></port>
      <backupHost></backupHost>
```

```
        <backupPort></backupPort>
        <peripheralId></peripheralId>
        <secure></secure>
    </cti>
</SystemConfig>
```

Note Any changes made to the settings through the SystemConfig API will require a Cisco Finesse Tomcat restart.

SystemConfig APIs

SystemConfig—Get

This API allows an administrator to get a copy of the SystemConfig object.

URI:	https://<FQDN>/finesse/api/SystemConfig
Example URI:	https://finesse1.xyz.com/finesse/api/SystemConfig
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 403: Forbidden 500: Internal Server Error

Example Response:	`<SystemConfig>`
	`   <uri>/finesse/api/SystemConfig</uri>`
	`   <cti>`
	`      <host>10.1.1.1</host>`
	`      <port>42027</port>`
	`      <backupHost>10.1.1.2</backupHost>`
	`      <backupPort>42027</backupPort>`
	`      <peripheralId>5000</peripheralId>`
	`      <secure>true</secure>`
	`   </cti>`
	`</SystemConfig>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

SystemConfig—Set

This API allows an administrator to configure the CTI server settings.

Note If you do not specify the backupHost and backupPort during a PUT operation but they were configured at an earlier time, the PUT operation removes these values from the database.

URI:	https://<FQDN>/finesse/api/SystemConfig
Example URI:	https://finesse1.xyz.com/finesse/api/SystemConfig
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<SystemConfig>`
	`   <uri>/finesse/api/SystemConfig</uri>`
	`   <cti>`
	`      <host>10.1.1.1</host>`
	`      <port>42027</port>`
	`      <backupHost>10.1.1.2</backupHost>`
	`      <backupPort>42027</backupPort>`
	`      <peripheralId>5000</peripheralId>`
	`      <secure>false</secure>`
	`   </cti>`
	`</SystemConfig>`

Request Parameters:	host (required): Hostname or IP address of the primary (A Side) CTI server Port (required): Port number of the primary (A Side) CTI server backupHost (required if backupPort is present): Hostname or IP address of the backup (B Side) CTI server backupPort (required if backupHost is present): Port number of the backup (B Side) CTI server peripheralId (required): ID of the CTI server peripheral secure (optional): enables secure encryption to configure secure CTI connection depending on value set to true or false. By default, the value if not provided, will be false

HTTP Response:	200: Success 400: Invalid Input 400: Parameter Missing 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Invalid Input</ErrorType>`
	`          <ErrorMessage>port</ErrorMessage>`
	`          <ErrorData>65536</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

SystemConfig API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the SystemConfig object.	—	
cti	Collection	Information about the CTI server settings.	—	
-->host	String	The hostname or IP address of the primary (A Side) CTI server.	—	No special characters allowed except "." and "-".
-->port	Integer	The port of the primary (A Side) CTI server.	1–65535 Default value: 42027	
-->peripheralId	Integer	The ID of the CTI server peripheral.	1–32767 Default value: 5000	
-->backupHost	String	The hostname or IP address of the (B Side) backup CTI server.	—	Must not be the same as the hostname or IP address of the primary (A Side) CTI server. No special characters allowed except "." and "-".
-->backupPort	Integer	The port of the backup (B Side) CTI server.	1–65535	

Parameter	Type	Description	Possible Values	Notes
-->secure	Boolean	To enable secure encryption.	true or false	When the value is set to true enables secure encryption and if the value is set to false disables secure encryption

SystemConfig API Errors

Status	Error Type	Description
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized.
400	Parameter Missing	A required parameter was not provided in the request. For example, if the backupPort is provided but the backupHost is missing.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

ConfigInfo

The ConfigInfo object is a container element that holds the Cisco Finesse configuration details.

The ConfigInfo object is structured as follows:

```
<ConfigInfo>
 <totalSkillGroups></totalSkillGroups>
 <totalSupervisors></totalSupervisors>
 <totalTeams></totalTeams>
 <totalUsers></totalUsers>
 <uri></uri>
 <versionInfo></versionInfo>
</ConfigInfo>
```

ConfigInfo APIs

ConfigInfo—Get

This API allows an administrator to get the following information:

- Product version with the COP information
- Total skillGroups
- Total supervisors
- Total teams
- Total users

URI:	https://<FQDN>/finesse/api/ConfigInfo
Example URI:	https://finesse1.xyz.com/finesse/api/ConfigInfo
Security Constraints:	Administrators, agents, and supervisors can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 403: Forbidden 500: Internal Server Error

Example Response:	`<ConfigInfo>`
	`    <totalSkillGroups>5</totalSkillGroups>`
	`    <totalSupervisors>11</totalSupervisors>`
	`    <totalTeams>53</totalTeams>`
	`    <totalUsers>48</totalUsers>`
	`    <uri>/finesse/api/ConfigInfo</uri>`
	`    <versionInfo>12.0.0.99200-263</versionInfo>`
	`</ConfigInfo>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

ConfigInfo API Parameters

Parameter	Type	Description	Possible Values	Notes
totalSkillGroups	Integer	The total number of skill groups.	—	—
totalSupervisors	Integer	The total number of supervisors.	—	—
totalTeams	Integer	The total number of teams.	—	—
totalUsers	Integer	The total number of users.	—	—
uri	String	The URI to get a new copy of the ConfigInfo object.	—	—
versionInfo	String	The product version with the COP information.	—	—

ConfigInfo API Errors

Status	Error Type	Description
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session).
403	Forbidden	The user attempted to run the API against the secondary Cisco Finesse server. Configuration APIs cannot be run against the secondary Cisco Finesse server.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

ECCVariableConfig

The ECCVariableConfig object is a container element that holds ECC variable configuration details for Unified CCE.

The ECCVariableConfig object is structured as follows:

```
<ECCVariableConfig>
    <ECCVariable>
        <name>user.microapp.override_cli</name>
        <length>40</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.play_data</name>
        <length>40</length>
    </ECCVariable>
    <ECCVariable>
```

```
        <name>user.cvp_server_info</name>
        <length>40</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.media.id</name>
        <length>40</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.media_server</name>
        <length>40</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.app_media_lib</name>
        <length>10</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.locale</name>
        <length>5</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.FromExtVXML</name>
        <length>60</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.caller_input</name>
        <length>40</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.error_code</name>
        <length>2</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.CourtesyCallbackEnabled</name>
        <length>1</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.charset</name>
        <length>3</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.UseVXMLParams</name>
        <length>1</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.input_type</name>
        <length>1</length>
    </ECCVariable>
    <ECCVariable>
        <name>user.microapp.ToExtVXML</name>
        <length>80</length>
    </ECCVariable>
</ECCVariableConfig>
```

ECCVariableConfig APIs

ECCVariableConfig—Get ECC Variable Configuration

This API allows a user to get a copy of the ECC variable configuration in Unified CCE.

URI:	https://<FQDN>/finesse/api/ECCVariableConfig

Example URI:	https://finesse1.xyz.com/finesse/api/ECCVariableConfig
Security Constraints:	Administrators, agents, and supervisors can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 403: Forbidden 500: Internal Server Error

Example Response:	`<ECCVariableConfig>`
	`    <ECCVariable>`
	`        <name>user.microapp.override_cli</name>`
	`        <length>40</length>`
	`    </ECCVariable>`
	`    <ECCVariable>`
	`        <name>user.microapp.play_data</name>`
	`        <length>40</length>`
	`    </ECCVariable>`
	`</ECCVariableConfig>`

ECCVariableConfig API Parameters

Parameter	Type	Description	Possible Values	Notes
name	String	The name of the ECC variable (both scalar and array).	—	—
length	Number	The length of the ECC variable.	—	The maximum payload per ECC variable (bytes) in Unified UCC is 210. The ECC variable limitation for Unified CCE can be defined in the CTI server. The maximum value is 210.

ECCVariableConfig API Errors

Status	Error Type	Description
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session).
403	Forbidden	The user attempted to run the API against the secondary Cisco Finesse server. Configuration APIs cannot be run against the secondary Cisco Finesse server.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

ClusterConfig

The ClusterConfig object is a container element that holds Finesse cluster configuration. This container supports the addition of a single, secondary Finesse node. After the secondary Finesse node is installed and ready, it becomes part of the cluster.

Note ClusterConfig APIs apply only to Finesse deployments with Unified CCE. Because you need not configure cluster settings for Unified CCX deployments, these APIs are not supported for Finesse with Unified CCX.

This feature also reports replication status. Replication status determines whether a user is allowed to or restricted from changing the value of the secondary node.

The Finesse server interacts with the VOS database to get and set information about the secondary node.

The ClusterConfig object is structured as follows:

```
<ClusterConfig>
   <uri>/finesse/api/ClusterConfig</uri>
   <secondaryNode>
      <host></host>
   </secondaryNode>
</ClusterConfig>
```

Note Any changes made to the settings through the ClusterConfig API will require a Cisco Finesse Tomcat restart.

ClusterConfig APIs

ClusterConfig—Get

This API allows an administrator to get a copy of the ClusterConfig object.

URI:	https://<FQDN>/finesse/api/ClusterConfig

Example URI:	https://finesse1.xyz.com/finesse/api/ClusterConfig
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 403: Forbidden 500: Internal Server Error

Example Response:	`<ClusterConfig>`
	`   <uri>/finesse/api/ClusterConfig</uri>`
	`   <secondaryNode>`
	`      <host>10.1.1.1</host>`
	`   </secondaryNode>`
	`</ClusterConfig>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

ClusterConfig—Set

This API allows an administrator to configure cluster settings for Finesse.

URI:	https://<FQDN>/finesse/api/ClusterConfig
Example URI:	https://finesse1.xyz.com/finesse/api/ClusterConfig
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<ClusterConfig>`
	`   <uri>/finesse/api/ClusterConfig</uri>`
	`   <secondaryNode>`
	`      <host>10.1.1.1</host>`
	`   </secondaryNode>`
	`</ClusterConfig>`

Request Parameters:	host (required): Hostname or IP address of the secondary Finesse server
HTTP Response:	200: Success 400: Invalid Input 400: Parameter Missing 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Invalid Input</ErrorType>`
	`          <ErrorMessage>host</ErrorMessage>`
	`          <ErrorData>10.1.1</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

ClusterConfig API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the ClusterConfig object.	—	
secondaryNode	Collection	Information about secondary Finesse node.	—	
-->host	String	The hostname or IP address of the secondary Finesse node.	—	No special characters allowed except "." and "-".

ClusterConfig API Errors

Status	Error Type	Description
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized.
400	Parameter Missing	A required parameter was not provided in the request.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).

Status	Error Type	Description
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

EnterpriseDatabaseConfig

The EnterpriseDatabaseConfig object is a container element that holds the properties required for Finesse to connect to the Administration & Data Server database (AWDB) for user authentication.

Note The EnterpriseDatabaseConfig APIs apply only to Finesse deployments with Unified CCE. Because these settings do not apply to Finesse deployments with Unified CCX, these APIs are not supported with Unified CCX.

The EnterpriseDatabaseConfig object is structured as follows:

```
<EnterpriseDatabaseConfig>
   <uri>/finesse/api/EnterpriseDatabaseConfig</uri>
   <host></host>
   <backupHost></backupHost>
   <port></port>
   <databaseName></databaseName>
   <domain></domain>
   <username></username>
   <password></password>
</EnterpriseDatabaseConfig>
```

Note Any changes made to the settings through the EnterpriseDatabaseConfig API will require a Cisco Finesse Tomcat restart.

EnterpriseDatabaseConfig APIs

EnterpriseDatabaseConfig—Get

This API allows an administrator to get a copy of the EnterpriseDatabaseConfig object.

URI:	https://<FQDN>/finesse/api/EnterpriseDatabaseConfig
Example URI:	https://finesse1.xyz.com/finesse/api/EnterpriseDatabaseConfig
Security Constraints:	Only administrators can use this API.

HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 403: Forbidden 500: Internal Server Error

Example Response:	`<EnterpriseDatabaseConfig>`
	`   <uri>/finesse/api/EnterpriseDatabaseConfig</uri>`
	`   <host>10.1.1.1</host>`
	`   <backupHost>10.1.1.2</backupHost>`
	`   <port>1433</port>`
	`   <databaseName>ucce8x_awdb</databaseName>`
	`   <domain>xyz.com</domain>`
	`   <username>Administrator</username>`
	`   <password>admin_password</password>`
	`</EnterpriseDatabaseConfig>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

EnterpriseDatabaseConfig—Set

This API allows an administrator to configure the enterprise database settings.

Note If you do not specify the backupHost during a PUT operation but it was configured at an earlier time, the PUT operation resets the value for this parameter to blank.

The URI for this API contains the query parameter override. This parameter is optional and can be set to true or false.

Certain errors returned by this API can be overridden. If an error can be overridden, it contains an override XML element within the body with a value of "true". If Finesse cannot connect to the Enterprise database with the supplied parameters, the following error is returned.

```
<ApiErrors>
 <ApiError>
  <ErrorType>Invalid Input</ErrorType>
  <ErrorMessage>Enterprise Database Connection Validation Failed</ErrorMessage>
  <ErrorData>Unable to authenticate against the primary enterprise database</ErrorData>
  <Overrideable>true</Overrideable>
 </ApiError>
```

```
</ApiErrors>
```

If this API is called with the query parameter override set to "true", the validation is skipped, the error is overridden, and the API continues to run.

URI:	https://<FQDN>/finesse/api/EnterpriseDatabaseConfig?override='<true|false>'
Example URI:	https://finesse1.xyz.com/finesse/api/EnterpriseDatabaseConfig?override='true'
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<EnterpriseDatabaseConfig>`
	`   <uri>/finesse/api/EnterpriseDatabaseConfig</uri>`
	`   <host>10.1.1.1</host>`
	`   <backupHost>10.1.1.2</backupHost>`
	`   <port>1433</port>`
	`   <databaseName>ucce8.x_awdb</databaseName>`
	`   <domain>example.com</domain>`
	`   <username>Admin</username>`
	`   <password>password</password>`
	`</EnterpriseDatabaseConfig>`

Request Parameters:	host (required): Hostname or IP address of the AWDB server backupHost (optional): Hostname or IP address of the backup AWDB server Port (required): Port number of the AWDB server databaseName (required): Name of the AWDB domain (optional): Domain of the AWDB username (required): Username to sign in to the AWDB. If there is a domain specified, this must be a domain user. Otherwise it must be an SQL user. password (required): Password to sign in to the AWDB
HTTP Response:	200: Success 400: Invalid Input 400: Parameter Missing 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Invalid Input</ErrorType>`
	`          <ErrorMessage>port</ErrorMessage>`
	`          <ErrorData>65536</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

EnterpriseDatabaseConfig API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the EnterpriseDatabaseConfig object.	—	
host	String	The hostname or IP address of the AWDB server.	—	No special characters allowed except "." and "-".
backupHost	String	The hostname or IP address of the backup AWDB server.	—	No special characters allowed except "." and "-".
port	Integer	The port of the AWDB server.	1–65535	
databaseName	String	The name of the AWDB (for example, *ucceinstance*_awdb).	—	
domain	String	The domain of the AWDB.	—	
username	String	The username required to sign in to the AWDB. If there is a domain specified, this must be a domain user. Otherwise it must be an SQL user.	—	
password	String	The password required to sign in to the AWDB.	—	

EnterpriseDatabaseConfig API Errors

Status	Error Type	Description
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized.
400	Parameter Missing	A required parameter was not provided in the request. For example, if the backupPort is provided but the backupHost is missing.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).

Status	Error Type	Description
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

LayoutConfig

The LayoutConfig object is a container element that holds the layout XML for the Finesse desktop. The layout XML defines how tabs, labels, columns, and gadgets appear on the Finesse agent and supervisor desktops.

When the desktop loads, Finesse reads the label for each tab and attempts to find it in the resource bundle (as a key). If Finesse finds the key, it displays in the value in the tab. If Finesse does not find the key, it displays the key as the default value for the tab.

The following example shows how the key mappings appear in the resource bundle for the Home and Manage Call tabs:

```
finesse.container.tabs.agent.homeLabel=Home
finesse.container.tabs.agent.manageCallLabel=Manage Call
finesse.container.tabs.supervisor.homeLabel=Home
finesse.container.tabs.supervisor.manageCallLabel=Manage Call
```

Note Gadgets that reside on the Finesse server can be specified by a relative path, as shown in the following example:

```
/desktop/gadgets/<gadgetname>.xml
```

Gadgets that are hosted on a server other than the Finesse server must be specified with a fully-qualified URL (absolute path), as shown in the following example:

```
http://server.com/<path to gadget>/<gadget name>.xml
```

The LayoutConfig object is structured as follows for Unified CCE:

```
<!--
*Note: When you upgrade, modify Custom Layout XML appropriately to utilize the benefits of
 new gadgets.
-->
<finesseLayout xmlns="http://www.cisco.com/vtg/finesse">
    <!--  DO NOT EDIT. The version number for the layout XML. -->
    <version>1250.03</version>
    <configs>
        <!-- The Title for the application which can be customized. -->
        <config key="title" value="Cisco Finesse"/>
       <!-- The following entries are examples of changing defaults for desktop properties.

        To change any property, uncomment the respective line and set the appropriate value.

       For more details on the properties that can be customized, refer to the Cisco Finesse
 Administration Guide.
        Note: The customized properties can only be set in the configs section and are not
 role-specific. -->
        <!-- <config key="enableDragDropAndResizeGadget" value="false"/> -->
        <!-- <config key="wrapUpCountDown" value="true"/> -->
        <!-- <config key="desktopChatAttachmentEnabled" value="true"/> -->
        <!-- <config key="forceWrapUp" value="true"/> -->
        <!-- The logo file for the application -->
        <!-- For detailed instructions on using custom icons for logos and tabs,
        please refer to the section "Customize Title and Logo in the Header"
        in the Finesse Administration Guide. -->
        <!-- <config key="logo" value="/3rdpartygadget/files/cisco_finext_logo.png"/>  -->
    </configs>
    <header>
        <!--  Please ensure that at least one gadget/component is present within every
headercolumn tag -->
  <leftAlignedColumns>
   <headercolumn width="300px">
    <component id="cd-logo">
     <url>/desktop/scripts/js/logo.js</url>
    </component>
   </headercolumn>
   <headercolumn width="230px">
    <component id="agent-voice-state">
     <url>/desktop/scripts/js/agentvoicestate.component.js</url>
    </component>
   </headercolumn>
   <headercolumn width="251px">
    <component id="nonvoice-state-menu">
     <url>/desktop/scripts/js/nonvoice-state-menu.component.js</url>
    </component>
   </headercolumn>

  </leftAlignedColumns>
  <rightAlignedColumns>
   <headercolumn width="50px">
                <component id="broadcastmessagepopover">
                    <url>/desktop/scripts/js/teammessage.component.js</url>
                </component>
            </headercolumn>
   <headercolumn width="50px">
                <component id="chat">
                    <url>/desktop/scripts/js/chat.component.js</url>
                </component>
            </headercolumn>
   <headercolumn width="50px">
    <component id="make-new-call-component">
     <url>/desktop/scripts/js/makenewcall.component.js</url>
    </component>
   </headercolumn>
   <headercolumn width="72px">
    <component id="identity-component">
     <url>/desktop/scripts/js/identity-component.js</url>
    </component>
   </headercolumn>
  </rightAlignedColumns>
 </header>
    <layout>
        <role>Agent</role>
        <page>
            <gadget>/desktop/scripts/js/callcontrol.js</gadget>
        </page>
        <tabs>
            <tab>
```

```
                <id>home</id>
                <icon>home</icon>
                <label>finesse.container.tabs.agent.homeLabel</label>
                <columns>
                    <column>
                        <gadgets>
                         <

                          <!-- The following gadget is for CloudCherry Customer Experience
 Journey.
                         If CloudCherry is onboarded successfully with all configurations,
 then replace the url
                           with the actual url obtained by exporting the Cisco Finesse
gadget from CloudCherry -->
       <!-- <gadget>/3rdpartygadget/files/CXService/CiscoCXJourneyGadget.xml</gadget> -->

                           <gadget>/desktop/scripts/js/queueStatistics.js</gadget>

                           <!--
                The following Gadgets are for LiveData.
                If you wish to show LiveData Reports, then do the following:
                    1) Uncomment each Gadget you wish to show.
                  2) Replace all instances of "my-cuic-server.com" with the Fully Qualified
 Domain Name of your Intelligence Center Server.
                    3) [OPTIONAL] Adjust the height of the gadget by changing the
"gadgetHeight" parameter.
                IMPORTANT NOTES:
                    - In order for these Gadgets to work, you must have performed all
documented pre-requisite steps.
                   - Do *NOT* change the viewId (unless you have built a custom report and
 know what you are doing).
                    - The "teamName" will be automatically replaced with the Team Name of
the User logged into Finesse (for Team-specific layouts).
-->
                           <!-- HTTPS Version of LiveData Gadgets -->
                           <!-- TEAM STATUS REPORTS: 1. Agent Default view (default), 2.
Agent Skill Group Default view -->
                           <!--
<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?gadgetHeight=310&amp;


viewId_1=99E6C8E210000141000000D80A0006C4&amp;filterId_1=agent.id=CL%20teamName&amp;

viewId_2=9AB7848B10000141000001C50A0006C4&amp;filterId_2=agent.id=CL%20teamName</gadget>
-->
                         <!-- QUEUE STATUS REPORTS: 1. Skill Group Default view (default),
 2. Skill Group Utilization view, 3. Precision Queue Default view, 4. Precision Queue
Utilization view -->
                           <!--
<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?
gadgetHeight=310&amp;viewId_1=B7371BE210000144000002870A0007C5&amp;filterId_1=skillGroup.id=CL%20teamName
&amp;viewId_2=9E760C8B1000014B0000005A0A0006C4&amp;filterId_2=skillGroup.id=CL%20teamName
&amp;viewId_3=B71A630C10000144000002480A0007C5&amp;filterId_3=precisionQueue.id=CL%20teamName&amp;
viewId_4=286B86F01000014C000005330A0006C4&amp;filterId_4=precisionQueue.id=CL%20teamName</gadget>
 -->
                        </gadgets>
                    </column>
                </columns>
            </tab>
            <tab>
                <id>myStatistics</id>
                <icon>column-chart</icon>
                <label>finesse.container.tabs.agent.myStatisticsLabel</label>
                <columns>
```

```
                        <column>
                            <gadgets>

<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?
gadgetHeight=150&amp;viewId=0B8D11317ED54A80B64F3AE28C5139E5&amp;
filterId=agentStats.id=CL%20teamName</gadget>
                            </gadgets>
                        </column>
                    </columns>
                </tab>
                <tab>
                    <id>myHistory</id>
                    <icon>history</icon>
                    <label>finesse.container.tabs.agent.myHistoryLabel</label>
                    <columns>
                        <column>
                            <!-- The following gadgets are used for viewing the call history
and state history of an agent. -->
                            <gadgets>

<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?
gadgetHeight=280&amp;viewId=5FA44C6F930C4A64A6775B21A17EED6A&amp;
filterId=agentTaskLog.id=CL%20teamName</gadget>

<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?
gadgetHeight=280&amp;viewId=56BC5CCE8C37467EA4D4EFA8371258BC&amp;
filterId=agentStateLog.id=CL%20teamName</gadget>
                            </gadgets>
                        </column>
                    </columns>
                </tab>
                <!--
The following Gadgets are for LiveData.
If you wish to show More LiveData Reports, then do the following:
    1) Uncomment each Gadget you wish to show.
    2) Replace all instances of "my-cuic-server.com" with the Fully Qualified Domain Name
of your Intelligence Center Server.
    3) [OPTIONAL] Adjust the height of the gadget by changing the "gadgetHeight" parameter.
IMPORTANT NOTES:
    - In order for these Gadgets to work, you must have performed all documented pre-requisite
 steps.
    - Do *NOT* change the viewId (unless you have built a custom report and know what you
are doing).
    - The "teamName" will be automatically replaced with the Team Name of the User logged
into Finesse (for Team-specific layouts).
-->
                <!-- If you are showing the "More Live Data Reports" tab, then also uncomment
this section.
                <tab>
                    <id>moreLiveDataReports</id>
                    <icon>reports-more</icon>
                    <label>finesse.container.tabs.agent.moreLiveDataReportsLabel</label>
                    <gadgets>
-->
                <!-- HTTPS Version of LiveData Gadgets -->
                <!-- AGENT REPORTS: 1. Agent Default view (default) -->
<!--
<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?gadgetHeight=310&amp;
viewId_1=99E6C8E210000141000000D80A0006C4&amp;filterId_1=agent.id=CL%20teamName</gadget>-->
<!-- AGENT SKILL GROUP REPORTS: 1. Agent Skill Group Default view (default) -->
<!--
<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?gadgetHeight=310&amp;
viewId_1=9AB7848B10000141000001C50A0006C4&amp;filterId_1=agent.id=CL%20teamName</gadget>-->
```

```
            <!-- QUEUE STATUS SKILL GROUP REPORTS: 1. Skill Group Default view (default),
2. Skill Group Utilization view -->
            <!--
<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?gadgetHeight=310&amp;
viewId_1=B7371BE210000144000002870A0007C5&amp;filterId_1=skillGroup.id=CL%20teamName&amp;
viewId_2=9E760C8B1000014B0000005A0A0006C4&amp;filterId_2=skillGroup.id=CL%20teamName</gadget>-->

            <!-- QUEUE STATUS PRECISION QUEUE REPORTS: 1. Precision Queue Default view
(default), 2. Precision Queue Utilization view -->
            <!--
<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?gadgetHeight=310&amp;
viewId_1=B71A630C10000144000002480A0007C5&amp;filterId_1=precisionQueue.id=CL%20teamName&amp;
viewId_2=286B86F01000014C000005330A0006C4&amp;filterId_2=precisionQueue.id=CL%20teamName</gadget>-->

            <!-- If you are showing the "more reports" tab, then uncomment this section
too.
                </gadgets>
            </tab>
            -->
        </tabs>
    </layout>
    <layout>
        <role>Supervisor</role>
        <page>
            <gadget>/desktop/scripts/js/callcontrol.js</gadget>
        </page>
        <tabs>
            <tab>
                <id>home</id>
                <icon>home</icon>
                <label>finesse.container.tabs.supervisor.homeLabel</label>
                <columns>
                    <column>
                        <gadgets>
                         <!-- The following gadget is for CloudCherry Customer Experience
Analytics.
                          If CloudCherry is onboarded successfully with all configurations,
 then replace the url
                            with the actual url obtained by exporting the Cisco Finesse
gadget from CloudCherry -->
      <!-- <gadget>/3rdpartygadget/files/CXService/CiscoCXAnalyticsGadget.xml</gadget> -->


                            <gadget
id="team-performance">/desktop/scripts/js/teamPerformance.js</gadget>
                           <!-- The following gadgets are used for viewing the call history
 and state history of an agent selected in the Team Performance Gadget. -->
                            <!-- The following gadgets are managed(loaded and displayed)
by the team performance gadget (associated with id "team-performance").
                                 This association is done using the mapping of managedBy
attribute of the managed gadgets, to the id of managing gadget.
                                 If the id for team performance gadget is changed, the
values for the associated managedBy attribute
                                 for the managed gadgets, also need to be updated with the
 new id.

                                 These managed gadgets are not displayed by default, but
would be displayed when the option
                                 "view history" is selected, for an agent, in the team
performance gadget.

                                 Note: As managed gadgets are not displayed by default,
placing managed gadgets alone on
                                 separate columns of their own, would display blank space
```

```
in that area.
                              For more details on managed gadgets and managedBy attribute,
 please refer to Finesse Administration Guide.
                            -->
<gadget
managedBy="team-performance">https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?
gadgetHeight=275&amp;viewId=630CB4C96B0045D9BFF295A49A0BA45E&amp;filterId=agentTaskLog.id=AgentEvent:Id
&amp;type=dynamic&amp;maxRows=20</gadget>
<gadget
managedBy="team-performance">https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?
gadgetHeight=275&amp;viewId=56BC5CCE8C37467EA4D4EFA8371258BC&amp;filterId=agentStateLog.id=AgentEvent:Id
&amp;type=dynamic&amp;maxRows=20</gadget>
                        </gadgets>
                    </column>
                </columns>
            </tab>
            <tab>
                <id>myHistory</id>
                <icon>history</icon>
                <label>finesse.container.tabs.agent.myHistoryLabel</label>
                <columns>
                    <column>
                        <!-- The following gadgets are used for viewing the call history
and state history of a logged in supervisor. -->
                        <gadgets>

<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?gadgetHeight=280&amp;
viewId=5FA44C6F930C4A64A6775B21A17EED6A&amp;filterId=agentTaskLog.id=CL%20teamName</gadget>


<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?gadgetHeight=280&amp;
viewId=56BC5CCE8C37467EA4D4EFA8371258BC&amp;filterId=agentStateLog.id=CL%20teamName</gadget>

                        </gadgets>
                    </column>
                </columns>
            </tab>
            <tab>
                <id>teamData</id>
                <icon>team-data</icon>
                <label>finesse.container.tabs.supervisor.teamDataLabel</label>
                <columns>
                    <column>
                        <!-- The following gadget is used by the supervisor to view an
agent's queue interval details. -->
                        <gadgets>

<gadget>https://my-cuic-server.com:8444/cuic/gadget/LiveData/LiveDataGadget.jsp?gadgetHeight=310&amp;
viewId=0B8D11317ED54A80B64F3AE28C5139E5&amp;filterId=agentStats.id=CL%20teamName</gadget>

<gadget>https://my-cuic-server.com:8444/cuic/gadget/Historical/HistoricalGadget.jsp?
viewId=BD9A8B7DBE714E7EB758A9D472F0E7DC&amp;linkType=htmlType&amp;viewType=Grid
&amp;refreshRate=900&amp;@start_date=RELDATE%20THISWEEK&amp;@end_date=RELDATE%20THISWEEK&amp;
@agent_list=CL%20~teams~&amp;gadgetHeight=360</gadget>
                        </gadgets>
                    </column>
                </columns>
            </tab>
            <tab>
                <id>queueData</id>
                <icon>storage</icon>
                <label>finesse.container.tabs.supervisor.queueDataLabel</label>
                <columns>
                    <column>
```

```
                        <gadgets>
                            <gadget>/desktop/scripts/js/queueStatistics.js</gadget>
                        </gadgets>
                    </column>
                </columns>
            </tab>

        </tabs>
    </layout>
</finesseLayout>
```

The LayoutConfig object is structured as follows for Unified CCX:

```
<!--
*Note:
 - When you upgrade, modify Custom Layout XML appropriately to utilize the benefits of new
 gadgets.
 - Remove the Agent State Log gadget from My Statistics tab, as it is available in the My
History tab.
-->
<finesseLayout xmlns="http://www.cisco.com/vtg/finesse">
    <!--  DO NOT EDIT. The version number for the layout XML. -->
    <version>1250.03</version>
    <configs>
        <!-- The Title for the application which can be customized. -->
        <config key="title" value="Cisco Finesse"/>
       <!-- The following entries are examples of changing defaults for desktop properties.

       To change any property, uncomment the respective line and set the appropriate value.

       For more details on the properties that can be customized, refer to the Cisco Finesse
 Administration Guide.
        Note: The customized properties can only be set in the configs section and are not
 role-specific. -->
        <!-- <config key="enableDragDropAndResizeGadget" value="false"/> -->
        <!-- <config key="wrapUpCountDown" value="true"/> -->
        <!-- <config key="desktopChatAttachmentEnabled" value="true"/> -->
        <!-- <config key="enableShortCutKeys" value="true"/> -->
        <!-- The logo file for the application -->
        <!-- For detailed instructions on using custom icons for logos and tabs,
        please refer to the section "Customize Title and Logo in the Header"
        in the Finesse Administration Guide. -->
        <!-- <config key="logo" value="/3rdpartygadget/files/cisco_finext_logo.png"/> -->
    </configs>
    <header>
        <!--  Please ensure that at least one gadget/component is present within every
headercolumn tag -->
        <leftAlignedColumns>
            <headercolumn width="300px">
                <component id="cd-logo">
                    <url>/desktop/scripts/js/logo.js</url>
                </component>
            </headercolumn>
            <headercolumn width="230px">
                <component id="agent-voice-state">
                    <url>/desktop/scripts/js/agentvoicestate.component.js</url>
                </component>
            </headercolumn>
            <headercolumn width="251px">
                <component id="nonvoice-state-menu">
                    <url>/desktop/scripts/js/nonvoice-state-menu.component.js</url>
                </component>
            </headercolumn>

        </leftAlignedColumns>
        <rightAlignedColumns>
         <headercolumn width="50px">
                <component id="broadcastmessagepopover">
                    <url>/desktop/scripts/js/teammessage.component.js</url>
                </component>
            </headercolumn>
         <headercolumn width="50px">
                <component id="chat">
                    <url>/desktop/scripts/js/chat.component.js</url>
                </component>
            </headercolumn>
            <headercolumn width="50px">
                <component id="make-new-call-component">
                    <url>/desktop/scripts/js/makenewcall.component.js</url>
                </component>
            </headercolumn>
            <headercolumn width="72px">
                <component id="identity-component">
                    <url>/desktop/scripts/js/identity-component.js</url>
                </component>
            </headercolumn>
        </rightAlignedColumns>
    </header>
    <layout>
        <role>Agent</role>
        <page>
            <gadget>/desktop/scripts/js/callcontrol.js</gadget>
            <!--
    The following Gadget is used for WebChat and Email. It is *ONLY* supported with WebChat
 and Email.  If you are not using WebChat and Email, then
   remove it. If you are using WebChat or Email, include this Gadget in the Desktop Layouts
 used by Teams associated with chat and email
    CSQs. To include this functionality:
        1) Remove these comments leaving the gadget

    RESTRICTIONS:
        - The NonVoiceControl gadget must be configured as a page level gadget
        - The NonVoiceControl gadget must not be configured in a column

            <gadget
hidden="true">https://localhost:8445/uccx-nvcontrol/gadgets/NonVoiceControl.xml</gadget>
-->
        </page>
        <tabs>
            <tab>
                <id>home</id>
                <icon>home</icon>
                <label>finesse.container.tabs.agent.homeLabel</label>
                <columns>
                    <column>
                        <gadgets>
       <!-- The following gadget is for CloudCherry Customer Experience Journey.
                          If CloudCherry is onboarded successfully with all configurations,
 then replace the url
                            with the actual url obtained by exporting the Cisco Finesse
gadget from CloudCherry -->
       <!-- <gadget>/3rdpartygadget/files/CXService/CiscoCXJourneyGadget.xml</gadget> -->


<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=310&
viewId=76D964AD10000140000000830A4E5E6F&filterId=AgentCSQStats.csqName=CL&compositeFilterId=
AgentCSQStats.AgentIds.agentId=loginId</gadget>

<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=310&
```

```
viewId=5C626F9C10000140000000600A4E5B33&filterId=ResourceIAQStats.resourceId=CL</gadget>
                        </gadgets>
                    </column>
                </columns>
            </tab>
            <tab>
                <id>myHistory</id>
                <icon>history</icon>
                <label>finesse.container.tabs.agent.myHistoryLabel</label>
                <columns>
                    <column>
                        <!-- The following gadgets are used for viewing the call history
and state history of an agent. -->
                        <gadgets>

<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=280&
viewId=ECD59EE071BE439A898187B29575E175&filterId=AgentCallLogDetailStats.agentID=loginId</gadget>


<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=280&
viewId=5D411E8A10000140000000230A4E5E6B&filterId=AgentStateDetailStats.agentID=loginId</gadget>

                        </gadgets>
                    </column>
                </columns>
            </tab>
            <tab>
                <id>myStatistics</id>
                <icon>column-chart</icon>
                <label>finesse.container.tabs.agent.myStatisticsLabel</label>
                <columns>
                    <column>
                        <gadgets>

<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=150&
viewId=67D4371110000140000001080A4E5E6B&filterId=ResourceIAQStats.resourceId=loginId</gadget>

                        </gadgets>
                    </column>
                </columns>
            </tab>
            <!--
    The following Tab and Gadget are used for WebChat and Email. They are *ONLY* supported
 with WebChat and Email. If you are not using WebChat or Email, then
    remove them. If you are using WebChat or Email, include this Gadget in the Desktop
Layouts used by Teams associated with chat or email
    CSQs.  To include this functionality:
        1) Remove these comments leaving the tab and gadget
        2) Replace all instances of "my-ccp-server" with the Fully Qualified Domain Name
of your CCP Server.
        3) [OPTIONAL] Adjust the height of the gadget by changing the "gadgetHeight"
parameter.

    IMPORTANT NOTE:
        - In order for this Gadget to work, you must have performed all documented
prerequisite steps.

    RESTRICTIONS:
        - The multisession-reply-gadget must not be configured as a page level gadget
        - The multisession-reply-gadget must not be configured in a column


            <tab>
                <id>manageNonVoiceMedia</id>
```

```
                    <icon>settings</icon>
                    <label>finesse.container.tabs.agent.manageNonVoiceMediaLabel</label>
                    <columns>
                        <column>
                            <gadgets>

<gadget>https://my-ccp-server/multisession/ui/gadgets/multisession-reply-gadget.xml?gadgetHeight=590</gadget>

                            </gadgets>
                        </column>
                    </columns>
                </tab>
-->
            </tabs>
        </layout>
        <layout>
            <role>Supervisor</role>
            <page>
                <gadget>/desktop/scripts/js/callcontrol.js</gadget>
                <!--
    The following Gadget is used for WebChat and Email.  It is *ONLY* supported with WebChat
 and Email.  If you are not using WebChat and Email, then
    remove it.  If you are using WebChat or Email, include this Gadget in the Desktop Layouts
 used by Teams associated with chat or email
    CSQs. To include this functionality:
        1) Remove these comments leaving the gadget

    RESTRICTIONS:
        - The NonVoiceControl gadget must be configured as a page level gadget
        - The NonVoiceControl gadget must not be configured in a column
        - The NonVoiceControl gadget is a headless gadget(i.e., with no display of its own),

              but has to be available for the agent's non-voice state control to be able
to
              set agent states for WebChat and Email.

                <gadget
hidden="true">https://localhost:8445/uccx-nvcontrol/gadgets/NonVoiceControl.xml</gadget>
-->
            </page>
            <tabs>
                <tab>
                    <id>manageTeam</id>
                    <icon>manage-team</icon>
                    <label>finesse.container.tabs.supervisor.manageTeamLabel</label>
                    <columns>
                        <column>
                            <gadgets>
                             <!-- The following gadget is for CloudCherry Customer Experience
Analytics.
                              If CloudCherry is onboarded successfully with all configurations,
 then replace the url
                                with the actual url obtained by exporting the Cisco Finesse
gadget from CloudCherry -->
      <!-- <gadget>/3rdpartygadget/files/CXService/CiscoCXAnalyticsGadget.xml</gadget> -->


                                <gadget
id="team-performance">/desktop/scripts/js/teamPerformance.js</gadget>
                               <!-- The following gadgets are used for viewing the call history
 and state history of an agent selected in the Team Performance Gadget. -->
                                <!-- The following gadgets are managed(loaded and displayed)
by the team performance gadget (associated with id "team-performance").
                                     This association is done using the mapping of managedBy
```

```
attribute of the managed gadgets, to the id of managing gadget.
                                 If the id for team performance gadget is changed, the
values for the associated managedBy attribute
                                 for the managed gadgets, also need to be updated with the
 new id.

                                 These managed gadgets are not displayed by default, but
would be displayed when the option
                                 "view history" is selected, for an agent, in the team
performance gadget.

                                 Note: As managed gadgets are not displayed by default,
placing managed gadgets alone on
                                 separate columns of their own, would display blank space
in that area.
                               For more details on managed gadgets and managedBy attribute,
 please refer to Finesse Administration Guide.
                            -->
                            <gadget
managedBy="team-performance">https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?
gadgetHeight=275&viewId=D6D0B6740B0040D5A089FD1C09F5C72C&filterId=
AgentCallLogDetailStats.agentID=AgentEvent:Id&type=dynamic&maxRows=20</gadget>
                            <gadget
managedBy="team-performance">https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?
gadgetHeight=275&viewId=5D411E8A10000140000000230A4E5E6B&filterId=
AgentStateDetailStats.agentID=AgentEvent:Id&type=dynamic&maxRows=20</gadget>
                        </gadgets>
                    </column>
                </columns>
            </tab>
            <tab>
                <id>myHistory</id>
                <icon>history</icon>
                <label>finesse.container.tabs.supervisor.myHistoryLabel</label>
                <columns>
                    <column>
                        <!-- The following gadgets are used for viewing the call history
and state history of a supervisor. -->
                        <gadgets>

<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=280&
viewId=ECD59EE071BE439A898187B29575E175&filterId=AgentCallLogDetailStats.agentID=loginId</gadget>


<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=280&
viewId=5D411E8A10000140000000230A4E5E6B&filterId=AgentStateDetailStats.agentID=loginId</gadget>

                        </gadgets>
                    </column>
                </columns>
            </tab>
            <tab>
                <id>teamData</id>
                <icon>team-data</icon>
                <label>finesse.container.tabs.supervisor.teamDataLabel</label>
                <columns>
                    <column>
                        <gadgets>

<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=620&
viewId_1=7291DCB410000140000000890A4E5B33&filterId_1=ResourceIAQStats.resourceId=CL&viewId_2=728283C210000140000000530A4E5B33
&filterId_2=ResourceIAQStats.resourceId=CL</gadget>
                            <!--
    The following Gadget is used for WebChat and Email.  It is *ONLY* supported with WebChat
 and Email.  If you are not using WebChat or Email, then
   remove it.  If you are using WebChat or Email, include this Gadget in the Desktop Layouts
 used by Teams associated with chat or email
    CSQs.  To include this functionality:
        1) Remove these comments leaving the gadget


<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=310&
viewId=F2F1FC17100001440000014E0A4E5D48&filterId=ChatAgentStats.agentId=CL</gadget>

<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=310&
viewId=BCC5767B1000014F000000580A4D3FA7&filterId=EmailAgentStats.agentId=CL</gadget>
 -->
                            <!--
    The following Gadgets are used for Predictive/Progressive/Preview Agent Outbound.
    To include this functionality:
    1) Remove these comments leaving the gadget


<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=310&
viewId_1=FD919FB9100001440000005D0A4E5B29&filterId_1=ResourceIAQStats.resourceId=CL&viewId_2=FD919FB510000144000000470A4E5B29
&filterId_2=ResourceIAQStats.resourceId=CL</gadget>
-->
                        </gadgets>
                    </column>
                </columns>
            </tab>
            <tab>
                <id>queueData</id>
                <icon>storage</icon>
                <label>finesse.container.tabs.supervisor.queueDataLabel</label>
                <columns>
                    <column>
                        <gadgets>

<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=620
&viewId_1=C8E2DB1610000140000000A60A4E5E6B&filterId_1=VoiceIAQStats.esd
Name=CL&viewId_2=9A7A14CE10000140000000ED0A4E5E6B&filterId_2=VoiceCSQDetailsStats.
agentId=CL&compositeFilterId=VoiceCSQDetailsStats.AgentVoiceCSQNames.agentVoiceCSQName=CL&
viewId_3=C8EF510810000140000000EB0A4E5E6B&filterId_3=VoiceIAQStats.esdName=CL&viewId_4=
C8EE2419100001400000000C30A4E5E6B&
filterId_4=VoiceIAQStats.esdName=CL</gadget>
                            <!--
    The following Gadget is used for WebChat and Email. It is *ONLY* supported with WebChat
 and Email.  If you are not using WebChat or Email, then
   remove it.  If you are using WebChat or Email, include this Gadget in the Desktop Layouts
 used by Teams associated with chat or email
    CSQs.  To include this functionality:
        1) Remove these comments leaving the gadget


<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=310&
viewId=E42ED788100001440000007B0A4E5CA1&filterId=ChatQueueStatistics.queueName=CL</gadget>

<gadget>https://localhost:8445/cuic/gadget/LiveData/LiveDataGadget.xml?gadgetHeight=310&
viewId=13970B4E100001500000021C0A4D3FA7&filterId=EmailQueueStatistics.queueName=CL</gadget>

 -->
                        </gadgets>
                    </column>
                </columns>
            </tab>
            <!--
    The following Tab and Gadget are used for WebChat and Email. They are *ONLY* supported
 with WebChat and Email. If you are not using WebChat or Email, then
    remove them. If you are using WebChat or Email, include this Gadget in the Desktop
Layouts used by Teams associated with chat or email
    CSQs.  To include this functionality:
        1) Remove these comments leaving the tab and gadget
        2) Replace all instances of "my-ccp-server" with the Fully Qualified Domain Name
of your CCP Server.
        3) [OPTIONAL] Adjust the height of the gadget by changing the "gadgetHeight"
parameter.

    IMPORTANT NOTE:
        - In order for this Gadget to work, you must have performed all documented
prerequisite steps.

    RESTRICTIONS:
        - The multisession-reply-gadget must not be configured as a page level gadget
        - The multisession-reply-gadget must not be configured in a column


            <tab>
                <id>manageNonVoiceMedia</id>
                <icon>settings</icon>
                <label>finesse.container.tabs.supervisor.manageNonVoiceMediaLabel</label>
                <columns>
                    <column>
                        <gadgets>

<gadget>https://my-ccp-server/multisession/ui/gadgets/multisession-reply-gadget.xml?gadgetHeight=590</gadget>

                        </gadgets>
                    </column>
                </columns>
            </tab>
-->
<!--
   The following gadget provides Supervisor with advanced capabilities.
   Using this gadget, supervisors can manage Queues, Prompts, Calendars, and so on.
   Before including this gadget in Desktop Layout,
   ensure that the advanced capability is enabled in Unified CCX Administration.

            <tab>
                <id>ASCGadget</id>
                <icon>admin</icon>
                <label>finesse.container.tabs.supervisor.advancedcapabilities</label>
                <columns>
                    <column>
                        <gadgets>

<gadget>https://localhost:8445/ascgadget/gadgets/ascgadget.xml</gadget>
                        </gadgets>
                    </column>
                </columns>
            </tab>
-->
        </tabs>
    </layout>
</finesseLayout>
```

LayoutConfig APIs

LayoutConfig—Get

This API allows an administrator to get a copy of the LayoutConfig object.

URI:	https://<FQDN>/finesse/api/LayoutConfig/default
Example URI:	https://finesse1.xyz.com/finesse/api/LayoutConfig/default
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 403: Forbidden 500: Internal Server Error

Example Response:	`<LayoutConfig>`
	`   <uri>/finesse/api/LayoutConfig/default</uri>`
	`      <layoutxml>`
	
	`         ...`
	
	`      </layoutxml>`
	`</LayoutConfig>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

LayoutConfig—Set

This API allows an administrator to update the default layout settings for the Finesse desktop.

Note The XML data is verified to ensure that it is valid XML and that it conforms to the Finesse schema.

URI:	https://<FQDN>/finesse/api/LayoutConfig/default

Example URI:	https://finesse1.xyz.com/finesse/api/LayoutConfig/default
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<LayoutConfig>`
	`   <layoutxml><?xml version="1.0" encoding="UTF-8">`
	`      ...`
	`   </layoutxml>`
	`</LayoutConfig>`

Request Parameters:	layoutxml (required): The XML data that determines the layout of the Finesse desktop
HTTP Response:	200: Success 400: Invalid Input 400: Parameter Missing 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Invalid Input</ErrorType>`
	`          <ErrorMessage>layoutxml</ErrorMessage>`
	`     </ApiError>`
	`</ApiErrors>`

LayoutConfig API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the LayoutConfig object.	—	
layoutxml	String	The XML data that determines the layout of the Finesse desktop.	—	Must be valid XML and must conform to the Finesse schema.

LayoutConfig API Errors

Status	Error Type	Description
400	Invalid Input	The submitted XML is invalid or does not conform to the Finesse schema.

Status	Error Type	Description
400	Parameter Missing	The layout XML file was not provided.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

ReasonCode

The ReasonCode object represents a reason code that can be applied when an agent changes state. There are two categories of reason codes: not ready reason codes and sign out reason codes.

Administrators can use either the ReasonCode APIs or the Finesse administration console to configure not ready and sign out reason codes. When using the APIs to configure reason codes, the administrator specifies the category of reason code in the request (NOT_READY or LOGOUT).

To prevent reporting problems, define your reason codes consistently on both Finesse and the platform (Unified CCE or Unified CCX). For example, if you create a not ready reason code in Finesse with a code of 413 and a label of “Meeting”, but create a not ready reason code in Unified CCE with a code of 413 and a description of “Lunch Break”, the Unified CCE report shows “Lunch Break” for any agent who selects that code. For more information about predefined reason codes for Unified CCE, see the *Cisco Unified Contact Center Enterprise Reporting User Guide* (http://www.cisco.com/en/US/products/sw/custcosw/ps1844/products_user_guide_list.html). For more information about predefined reason codes for Unified CCX, see the *Cisco Unified Contact Center Express CTI Protocol Developer Guide*.

Note System reason codes are defined by Unified CCE and Unified CCX. These reason codes are used by Finesse but not listed in the ReasonCode APIs.

The ReasonCode object is structured as follows:

```
<ReasonCode>
   <uri>/finesse/api/ReasonCode/{id}</uri>
   <category>NOT_READY|LOGOUT</category>
   <code></code>
   <label></label>
   <forAll>true|false</forAll>
   <systemCode>true|false</systemCode>
</ReasonCode>
```

ReasonCode APIs

ReasonCode—Get

The following GET APIs allow an administrator or an agent to get a copy of the ReasonCode object.

URI:	https://<FQDN>/finesse/api/ReasonCode/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/ReasonCode/45
Security Constraints:	Administrators and agents can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<ReasonCode>`
	`   <uri>/finesse/api/ReasonCode/45</uri>`
	`   <category>NOT_READY</category>`
	`   <code>10</code>`
	`   <label>Team Meeting</label>`
	`   <forAll>true</forAll>`
	`   <systemCode>true</systemCode>`
	`</ReasonCode>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

URI:	https://<FQDN>/finesse/api/ReasonCode?category=NOT_READY|LOGOUT&code=<code>
Example URI:	https://finesse1.xyz.com/finesse/api/ReasonCode?category=NOT_READY&code=45
Security Constraints:	Administrators and agents can use this API.
HTTP Method:	GET

Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<ReasonCode>`
	`   <uri>/finesse/api/ReasonCode/45</uri>`
	`   <category>NOT_READY</category>`
	`   <code>10</code>`
	`   <label>Team Meeting</label>`
	`   <forAll>true</forAll>`
	`   <systemCode>true</systemCode>`
	`</ReasonCode>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

ReasonCode—Get List

This API allows an administrator to get a list of not ready or sign out reason codes. The required URI parameter *category* specifies whether to retrieve not ready reason codes, sign out reason codes or both. If the category parameter is missing, the API returns an error.

URI:	https://<FQDN>/finesse/api/ReasonCodes?category=NOT_READY|LOGOUT|ALL
Example URI:	https://finesse1.xyz.com/finesse/api/ReasonCodes?category=ALL
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Invalid Input 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<ReasonCodes category="ALL">`
	`    <ReasonCode>`
	`        <uri>/finesse/api/ReasonCode/1</uri>`
	`          <category>NOT_READY</category>`
	`            ... Rest of ReasonCode Object ..`
	`    </ReasonCode>`
	`    <ReasonCode>`
	`        <uri>/finesse/api/ReasonCode/2</uri>`
	`          <category>LOGOUT</category>`
	`            ... Rest of ReasonCode Object ...`
	`    </ReasonCode>`
	`</ReasonCodes>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

ReasonCode—Create

This API allows an administrator to create a new reason code. The administrator specifies the category, code, label, and forAll attributes for the reason code.

Finesse supports a maximum of 100 global reason codes and 100 non-global reason codes for each category. You can create up to 100 global and 100 non-global reason codes with a category of NOT_READY, and 100 global and 100 non-global reason codes with a category of LOGOUT.

The forAll parameter determines if a reason code is global (true) or non-global (false).

Note If you provide two or more duplicate tags in the XML body for a POST operation, the value of the last duplicate tag is processed and all other duplicate tags are ignored.

URI:	https://<FQDN>/finesse/api/ReasonCode/
Example URI:	https://finesse1.xyz.com/finesse/api/ReasonCode/
Security Constraints:	Only administrators can use this API.
HTTP Method:	POST

Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<ReasonCode>`
	`   <category>NOT_READY</category>`
	`   <code>24</code>`
	`   <label>Lunch</label>`
	`   <forAll>true</forAll>`
	`</ReasonCode>`

Request Parameters:	category (required): The category of reason code (NOT_READY or LOGOUT) code (required):The code for the reason code label (required): The UI label for the reason code forAll (required): Whether the reason code is global (true) or non-global (false)
HTTP Response:	200: Success **Note** Finesse successfully created the new ReasonCode. The response contains an empty response body, and a "location:" header denoting the absolute URL of the newly created ReasonCode object 400: Bad Request 400: Finesse API Error 400: Maximum Exceeded 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

ReasonCode—Update

This API allows an administrator to modify an existing reason code. The administrator specifies an existing reason code via the uri, which includes its id, along with the value of the field to update.

At least one of the following parameters must be present in the HTTP request to update a reason code: code, label, or forAll. If none of these parameters are present, Finesse returns an Invalid Input error.

You do not need to include the attributes (code, label, or forAll) that you do not want to change. For example, if you want to change only the label for an existing reason code from "In Meeting" to "Attend Meeting", you can send the following request:

```
<ReasonCode>
    <label>Attend Meeting</label>
</ReasonCode>
```

Note If you provide two or more duplicate tags in the XML body for a PUT operation, the value of the last duplicate tag is processed and all other duplicate tags are ignored.

URI:	https://<FQDN>/finesse/api/ReasonCode/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/ReasonCode/456
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<ReasonCode>`
	`   <code>101</code>`
	`   <label>Lunch Break</label>`
	`   <forAll>true</forAll>`
	`</ReasonCode>`

Request Parameters:	id (required): The database ID for the reason code code:The code for the reason code label: The UI label for the reason code forAll: Whether the reason code is global (true) or non-global (false) **Note** Your request must include at least one of the following parameters: code, label, or forAll.
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

ReasonCode—Delete

This API allows an administrator to delete an existing reason code.

URI:	https://<FQDN>/finesse/api/ReasonCode/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/ReasonCode/ 423
Security Constraints:	Only administrators can use this API.
HTTP Method:	DELETE
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

ReasonCode API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the ReasonCode object.	—	
category	String	The category of the reason code.	NOT_READY, LOGOUT	
code	Integer	The code for the reason code	Unified CCE: 1–65535 Unified CCX: 1–999	The combination of code and category must be unique.

Parameter	Type	Description	Possible Values	Notes
label	String	The UI label for the reason code.	—	Maximum of 40 characters. The combination of label and category must be unique.
forAll	Boolean	Whether a reason code is global (true) or non-global (false).	true, false	
systemCode	Boolean	The reserved status of the reason code.	true, false	

ReasonCode API Errors

Status	Error Type	Description
400	Bad Request	One of the required parameters was not provided or is invalid
400	Finesse API Error	API error such as duplicated reason code or the reason code does not exist.
400	Maximum Exceeded	The maximum number of items has been exceeded.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
401	Invalid Authorization User Specified	The authenticated user tried to use the identity of another user.
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
404	Not Found	The specified resource cannot be found.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

WrapUpReason

The WrapUpReason object represents a reason that an agent can apply to a call during call wrap-up.

The WrapUpReason object is structured as follows:

```
<WrapUpReason>
   <uri>/finesse/api/WrapUpReason/{id}</uri>
   <label></label>
   <forAll>true|false</forAll>
</WrapUpReason>
```

WrapUpReason APIs

WrapUpReason—Get

This API allows an administrator to get a copy of the WrapUpReason object.

URI:	https://<FQDN>/finesse/api/WrapUpReason/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/WrapUpReason/31
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<WrapUpReason>`
	`   <uri>/finesse/api/WrapUpReason/31</uri>`
	`   <label>Product Question</label>`
	`   <forAll>false</forAll>`
	`</WrapUpReason>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

WrapUpReason—Get List

This API allows an administrator to get a list of wrap-up reasons.

URI:	https://<FQDN>/finesse/api/WrapUpReasons
Example URI:	https://finesse1.xyz.com/finesse/api/WrapUpReasons
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<WrapUpReasons>`
	`    <WrapUpReason>`
	`        ... Full WrapUpReason Object ...`
	`    </WrapUpReason>`
	`    <WrapUpReason>`
	`        ... Full WrapUpReason Object ...`
	`    </WrapUpReason>`
	`    <WrapUpReason>`
	`        ... Full WrapUpReason Object ...`
	`    </WrapUpReason>`
	`</WrapUpReasons>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

WrapUpReason—Create

This API allows an administrator to create a new wrap-up reason. The administrator specifies the label and forAll attributes for the wrap-up reason.

Note Cisco Finesse does not support the use of extended ASCII characters required for additional alphabets in the wrap-up reasons. You must use only ASCII characters in the 0-127 range. For example, if you add a wrap-up reason that contains the character à (ASCII 133), it does not appear correctly on the agent desktop.

Finesse supports a maximum of 100 global wrap-up reasons and 1500 non-global wrap-up reasons, for each category, with the restriction that a maximum of 100 non-global wrap-up reasons can be assigned to a single team.

The forAll parameter determines if a reason code is global (true) or non-global (false).

Note If you provide two or more duplicate tags in the XML body for a POST operation, the value of the last duplicate tag is processed and all other duplicate tags are ignored.

URI:	https://<FQDN>/finesse/api/WrapUpReason/
Example URI:	https://finesse1.xyz.com/finesse/api/WrapUpReason/
Security Constraints:	Only administrators can use this API.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<WrapUpReason>`
	`   <label>Recommendation</label>`
	`   <forAll>true</forAll>`
	`</WrapUpReason>`

Request Parameters:	label (required): The UI label for the wrap-up reason forAll (required): Whether the wrap-up reason is global (true) or non-global (false)
HTTP Response:	200: Success **Note** Finesse successfully created the new WrapUpReason. The response contains an empty response body, and a "location:" header denoting the absolute URL of the newly created WrapUpReason object 400: Maximum Exceeded 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

WrapUpReason—Update

This API allows an administrator to modify an existing wrap-up reason. The administrator references the wrap-up reason by its ID and specifies the values of the fields to update.

At least one of the following parameters must be present in the HTTP request to update a wrap-up reason: label or forAll. If neither of these parameters is present, Finesse returns an Invalid Input error.

You do not need to include the attributes (label or forAll) that you do not need to change. For example, if you want to change only the label for an existing reason code from "Wrong Number" to "Wrong Department", you can send the following request:

```
<WrapUpReason>
    <label>Wrong Department</label>
</WrapUpReason>
```

Note If you provide two or more duplicate tags in the XML body for a PUT operation, the value of the last duplicate tag is processed and all other duplicate tags are ignored.

URI:	https://<FQDN>/finesse/api/WrapUpReason/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/WrapUpReason/43
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<WrapUpReason>`
	`   <label>Sales Call</label>`
	`   <forAll>true</forAll>`
	`</WrapUpReason>`

Request Parameters:	id (required): The database ID for the wrap-up reason label (required): The UI label for the reason code forAll (required): Whether the reason code is global (true) or non-global (false)

HTTP Response:	200: Success 400: Finesse API Error 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	     `<ApiError>`
	          `<ErrorType>Authorization Failure</ErrorType>`
	          `<ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	          `<ErrorData>jsmith</ErrorData>`
	     `</ApiError>`
	`</ApiErrors>`

WrapUpReason—Delete

This API allows an administrator to delete an existing wrap-up reason.

URI:	https://<FQDN>/finesse/api/WrapUpReason/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/WrapUpReason/23
Security Constraints:	Only administrators can use this API.
HTTP Method:	DELETE
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	     `<ApiError>`
	          `<ErrorType>Authorization Failure</ErrorType>`
	          `<ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	          `<ErrorData>jsmith</ErrorData>`
	     `</ApiError>`
	`</ApiErrors>`

WrapUpReason API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the WrapUpReason object.	—	
label	String	The UI label for the wrap-up reason.	—	Maximum of 39 bytes (which is equal to 39 US English characters). The label must be unique.
forAll	Boolean	Whether a wrap-up reason is global (true) or non-global (false).	true, false	

WrapUpReason API Errors

Status	Error Type	Description
400	Bad Request	The request body is invalid
400	Finesse API Error	API error such as duplicated wrap-up reason or the wrap-up reason does not exist.
400	Maximum Exceeded	The maximum number of items has been exceeded.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
401	Invalid Authorization User Specified	The authenticated user tried to use the identity of another user.
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
404	Not Found	The specified resource cannot be found.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

ChatConfig

The ChatConfig object is a container element that holds the Finesse chat configuration and URLs of the primary and secondary chat servers.

The ChatConfig object is structured as follows:

```
<ChatConfig>
   <uri>/finesse/api/ChatConfig</uri>
   <primaryNode></primaryNode>
   <secondaryNode></secondaryNode>
</ChatConfig>
```

ChatConfig APIs

ChatConfig—Get

This API allows an administrator to get a copy of the ChatConfig object.

URI:	https://<FQDN>/finesse/api/ChatConfig
Example URI:	https://finesse1.xyz.com/finesse/api/ChatConfig
Security Constraints:	Administrators and agents can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 403: Forbidden 500: Internal Server Error

Example Response:	`<ChatConfig>`
	`    <primaryNode></primaryNode>`
	`    <secondaryNode></secondaryNode> <uri></uri>`
	`</ChatConfig>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

ChatConfig—Set

This API allows an administrator to configure the desktop chat server settings.

URI:	https://<FQDN>/finesse/api/ChatConfig
Example URI:	https://finesse1.xyz.com/finesse/api/ChatConfig
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<ChatConfig>`
	`   <primaryNode>https://finessecup.xyz.com/httpbinding</primaryNode>`
	
	`<secondaryNode>https://finessecup2.xyz.com/httpbinding</secondaryNode>`
	`</ChatConfig>`

Request Parameters:	primaryNode (optional): Primary node of the desktop chat server. secondaryNode (optional): The secondary node of the desktop chat server.
HTTP Response:	200: Success 400: Invalid Input 400: Parameter Missing 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>Invalid Input</ErrorType>`
	`        <ErrorData>primaryNode</ErrorData>`
	`        <ErrorMessage>Invalid Primary Node specified for`
	`ChatConfig</ErrorMessage>`
	`    </ApiError>`
	`</ApiErrors>`

ChatConfig API Parameters

Parameter	Type	Description	Possible Values	Notes
primaryNode	String	The primary server node of the chat server.	Valid URL with https protocol.	—
secondaryNode	String	The secondary server node of the chat server.	Valid URL with https protocol.	—

ChatConfig API Errors

Status	Error Type	Description
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized.
400	Parameter Missing	A required parameter was not provided in the request.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
403	Forbidden	The user is not authorized to use the API (the user is not an administrator).
500	Internal Server Error	Any runtime exception is caught and responded with this error.

Cloud Connect

Cloud Connect Configuration

Cloud Connect is a component that hosts services that allow customers to use cloud capabilities such as Cisco Webex Experience Management. The administrator can configure the Cloud Connect server settings in the Finesse administration console to contact the Cisco cloud services.

For more information, see *Cisco Unified Contact Center Enterprise Features Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/unified-contact-center-enterprise/products-feature-guides-list.html.

The Cloud Connect configuration object is structured as follows:

```
<CloudConnectConfig>
    <publisherAddress>ccpub.cisco.com</publisherAddress>
    <subscriberAddress>ccsub.cisco.com</subscriberAddress>
    <userName>admin</userName>
    <password>password</password>
</CloudConnectConfig>
```

Cloud Connect Configuration APIs

Cloud Connect Configuration—Get

This API allows an administrator to get a copy of the Cloud Connect configuration details. These details are stored in Finesse database.

URI:	https://<FQDN>/finesse/api/CloudConnectConfig
Example URI:	https://finesse1.xyz.com/finesse/api/CloudConnectConfig
Security Constraints:	Only administrators can use this API.

HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
Request Parameters:	—
HTTP Response:	200: Success 400: Bad Request 401: Authorization Failure 500: Internal Server Error 503: Service Unavailable

Example Response:	`<CloudConnectConfig>`
	`    <publisherAddress>ccpub.cisco.com</publisherAddress>`
	`    <subscriberAddress>ccsub.cisco.com</subscriberAddress>`
	`    <userName>admin</userName>`
	`    <password>password</password>`
	`</CloudConnectConfig>`

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>Authorization Failure</ErrorType>`
	`        <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`        <ErrorData>jsmith</ErrorData>`
	`    </ApiError>`
	`</ApiErrors>`

Cloud Connect Configuration—Set

This API allows an administrator to configure the Cloud Connect server settings for Finesse.

URI:	https://<FQDN>/finesse/api/CloudConnectConfig
Example URI:	https://finesse1.xyz.com/finesse/api/CloudConnectConfig
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<CloudConnectConfig>`
	`    <publisherAddress>ccpub.cisco.com</publisherAddress>`
	`    <subscriberAddress>ccsub.cisco.com</subscriberAddress>`
	`    <userName>admin</userName>`
	`    <password>password</password>`
	`</CloudConnectConfig>`

Request Parameters:	publisherAddress (required): The Cloud Connect publisher address. subscriberAddress (optional): The Cloud Connect subscriber address. userName (required): The username to invoke the Cloud Connect APIs. password (required): The password to invoke the Cloud Connect APIs.
HTTP Response:	200: Success 400: Bad Request 400: Parameter Missing 401: Authorization Failure 500: Internal Server Error 503: Service Unavailable

Example Response:	`<CloudConnectConfig>`
	`    <publisherAddress>ccpub.cisco.com</publisherAddress>`
	`    <subscriberAddress>ccsub.cisco.com</subscriberAddress>`
	`    <userName>admin</userName>`
	`    <password>********</password>`
	`</CloudConnectConfig>`

Example Failure Response:	**Example:**
	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>Authorization Failure</ErrorType>`
	`        <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`        <ErrorData>jsmith</ErrorData>`
	`    </ApiError>`
	`</ApiErrors>`
	Example:
	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>Parameter Missing</ErrorType>`
	`        <ErrorData>Username</ErrorData>`
	`        <ErrorMessage>Missing required parameter</ErrorMessage>`
	`    </ApiError>`
	`</ApiErrors>`

Cloud Connect Integration—Delete

This API allows an administrator to delete the Cloud Connect server settings for Finesse.

URI:	https://<FQDN>/finesse/api/CloudConnectConfig
Example URI:	https://finesse1.xyz.com/finesse/api/CloudConnectConfig
Security Constraints:	Only administrators can use this API.
HTTP Method:	DELETE
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 401: Authorization Failure 500: Internal Server Error 503: Service Unavailable

Example Failure Response:	`<ApiErrors>`
	`    <ApiError>`
	`        <ErrorType>Authorization Failure</ErrorType>`
	`        <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`        <ErrorData>jsmith</ErrorData>`
	`    </ApiError>`
	`</ApiErrors>`

Cloud Connect Configuration Parameters

Parameter	Type	Description	Possible Values	Notes
publisherAddress	String	The FQDN of the Cloud Connect publisher.	—	Mandatory
subscriberAddress	String	The FQDN of the Cloud Connect subscriber.	—	Optional
userName	String	The username to invoke the Cloud Connect APIs.	—	Mandatory
password	String	The password to invoke the Cloud Connect APIs.	—	Mandatory

Cloud Connect Configuration API Errors

Status	Error Type	Description
400	Bad Request	The request is malformed or incomplete or the extension provided is invalid.
400	Parameter Missing	The required parameter was not provided in the request.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (for example, an agent tries to use an API that only the administrator is authorized to use).
500	Internal Server Error	Any runtime exception is caught and responded with this error.

Status	Error Type	Description
503	Service Unavailable	A dependent service is down (for example, the Cisco Finesse Notification Service or Cisco Finesse Database). Finesse is OUT_OF_SERVICE.

Cloud Connect Services

Cloud Connect Token APIs provide a standard way to fetch the token from the CloudConnect server. These tokens are used by the Webex Experience Management, and Agent Answers gadgets to make API request from Cisco Finesse browser to the respective hosts or applications.

Cloud Connect Services APIs

Cloud Connect Services Token—Get

The Cloud Connect Services Token—Get API fetches the token from the CloudConnect server.

Tokens are fetched from the CloudConnect server based on the tokenSource and scopes parameters. At least one of the two parameters is mandatory and when both the parameters are provided, the scopes parameter is ignored.

The tokenSource parameter value can be either of cherrypoint or evapoint. Based on the value, the respective token request is made to CloudConnect to fetch the token from CloudCherry and Voicea applications respectively.

When tokenSource parameter is not provided, the scopes parameter value is used to request the Cloud Integration token from CloudConnect.

URI:	https://<FQDN>/finesse/api/CloudTokenService? tokenSource=<>&scopes=<>
Example URI:	https://finesse1.xyz.com:8445/finesse/api/CloudTokenService?tokenSource=cherrypoint https://finesse1.xyz.com:8445/finesse/api/CloudTokenService?scopes=cjp-analyzer:read
Security Constraints:	Administrators, agents, and supervisors can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	JSON
HTTP Request:	—
Request Parameters:	tokenSource(optional): The source where the CloudConnect tokens are fetched. Possible values: cherrypoint or evapoint. scopes (optional): Scopes of the fetched token. **Note** At least one parameter is required.

HTTP Response:	200: Success 400: Bad Request 401: Authorization Failure 500: Internal Server Error 503: Service Unavailable
Example Response:	`{ "access_token": "MjQxZDI5f-bafb-4d68" "expires_at": 161629128912891289128912 "token_type": "Bearer" }`
Example Failure Response:	`<ApiErrors> <ApiError> <ErrorType>Authorization Failure</ErrorType> <ErrorMessage>UNAUTHORIZED</ErrorMessage> <ErrorData>ACCESSTOKEN</ErrorData> </ApiError> </ApiErrors>`

Cloud Connect Services API Parameters

Parameter	Type	Description	Possible Values	Notes
tokenSource	String	The source from where the CloudConnect server tokens are fetched.	cherrypoint or evapoint	—
scopes	String	Scope of the fetched token.	—	—

Cloud Connect Management Service Config—Get

The Cloud Connect Services Management—Get API fetches the organization configuration details with which cloud connect is on-boarded.

URI:	https://<FQDN>/finesse/api/CloudConnectMgmtService/Config
Example URI:	https://finesse1.xyz.com:8445/finesse/api/CloudConnectMgmtService/Config
Security Constraints:	Administrators, agents, and supervisors can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	JSON
HTTP Request:	—
Request Parameters:	—

HTTP Response:	200: Success 400: Bad Request 401: Authorization Failure 500: Internal Server Error 503: Service Unavailable
Example Response:	`{ "u2cHost":"ciscospark.com", "orgId":"ccbu" }`
Example Failure Response:	`<ApiErrors><ApiError><ErrorType>Authorization Failure</ErrorType><ErrorMessage>UNAUTHORIZED</ErrorMessage><ErrorData>Jsmith</ErrorData></ApiError></ApiErrors>`

Cloud Connect Services API Errors

Status	Error Type	Description
400	Bad Request	The request is malformed or incomplete or the extension provided is invalid.
401	Authorization Failure	Authorization failed.
500	Internal Server Error	Any runtime exception is caught and responded with this error.
503	Service Unavailable	The dependent service is down (for example, the Cisco Finesse Notification Service or Cisco Finesse Database). Finesse is OUT_OF_SERVICE.

MediaPropertiesLayout

The MediaPropertiesLayout object represents the appearance of media properties in the call control gadget on the agent or supervisor desktop. Media properties are carried in Dialog objects. Administrators can create and customize multiple layouts for media properties.

The MediaPropertiesLayout supports callVariable1 through callVariable10, ECC variables, and the following blended agent (outbound) variables:

- BACampaign
- BAAccountNumber
- BAResponse
- BAStatus

- BADialedListID
- BATimeZone
- BABuddyName
- BACustomerNumber (Unified CCX only)

The MediaPropertiesLayout object is structured as follows:

```
<MediaPropertiesLayout>
    <uri>/finesse/api/MediaPropertiesLayout/{id}</uri>
    <name>Layout name</name>
    <description>Layout description</description>
    <type>DEFAULT|CUSTOM</type>
    <header>
        <entry>
            <displayName>Customer Name</displayName>
            <mediaProperty>callVariable1</mediaProperty>
            <showInPopOver>false</showInPopOver>
            <uiEditable>false</uiEditable>
        </entry>
    </header>
    <column>
        <entry>
            <displayName>Customer Name</displayName>
            <mediaProperty>callVariable1</mediaProperty>
            <showInPopOver>false</showInPopOver>
            <uiEditable>false</uiEditable>
        </entry>
        <entry>
            <displayName>Customer Acct#</displayName>
            <mediaProperty>user.cisco.acctnum</mediaProperty>
            <showInPopOver>false</showInPopOver>
            <uiEditable>false</uiEditable>
        </entry>
    </column>
    <column>
        <entry>
            <displayName>Support contract</displayName>
            <mediaProperty>callVariable2</mediaProperty>
            <showInPopOver>false</showInPopOver>
            <uiEditable>false</uiEditable>
        </entry>
        <entry>
            <displayName>Product calling about</displayName>
            <mediaProperty>callVariable3</mediaProperty>
            <showInPopOver>false</showInPopOver>
            <uiEditable>false</uiEditable>
        </entry>
    </column>
</MediaPropertiesLayout>
```

MediaPropertiesLayout APIs

MediaPropertiesLayout—Get

This API allows an administrator to get a copy of the media properties layout associated with the specified ID.

URI:	https://<FQDN>/finesse/api/MediaPropertiesLayout/{id}
Example URI:	https://finesse1.xyz.com/finesse/api/MediaPropertiesLayout/15
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Response:	`<MediaPropertiesLayout>`
	`... Full MediaPropertiesLayoutConfig Object ...`
	`</MediaPropertiesLayout>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

MediaPropertiesLayout—Get Default Layout

This API allows an administrator to get a copy of the default MediaPropertiesLayout object.

Note Cisco Finesse supports this API for backward compatibility, but to get the default layout, developers must specify the default MediaPropertiesLayout ID in the MediaPropertiesLayout—Get API.

URI:	https://<FQDN>/finesse/api/MediaPropertiesLayout/default
Example URI:	https://finesse1.xyz.com/finesse/api/MediaPropertiesLayout/default
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—

Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Response:	`<MediaPropertiesLayout>`
	`    <uri>/finesse/api/MediaPropertiesLayout/{id}</uri>`
	`    <name>Default</name>`
	`    <description>This is the default layout</description>`
	`    <type>DEFAULT</type>`
	`    <header>`
	`        <entry>`
	`            <displayName>Customer Name</displayName>`
	`            <mediaProperty>callVariable1</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`    </header>`
	`    <column>`
	`        <entry>`
	`            <displayName>Customer Name</displayName>`
	`            <mediaProperty>callVariable1</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`        <entry>`
	`            <displayName>Customer Acct#</displayName>`
	`            <mediaProperty>user.cisco.acctnum</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`    </column>`
	`    <column>`
	`        <entry>`
	`            <displayName>Support contract</displayName>`
	`            <mediaProperty>callVariable2</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`        <entry>`
	`            <displayName>Product calling about</displayName>`
	`            <mediaProperty>callVariable3</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`    </column>`
	`</MediaPropertiesLayout>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

MediaPropertiesLayout—Get List

This API allows an administrator to list all the media properties layouts configured in the system.

URI:	https://<FQDN>/finesse/api/MediaPropertiesLayouts
Example URI:	https://finesse1.xyz.com/finesse/api/MediaPropertiesLayouts
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API error 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 500: Internal Server Error

Example Response:	`<MediaPropertiesLayouts>`
	`    <MediaPropertiesLayout>`
	`        ... Full MediaPropertiesLayout Object ...`
	`    </MediaPropertiesLayout>`
	`    <MediaPropertiesLayout>`
	`        ... Full MediaPropertiesLayout Object ...`
	`    </MediaPropertiesLayout>`
	`    <MediaPropertiesLayout>`
	`        ... Full MediaPropertiesLayout Object ...`
	`    </MediaPropertiesLayout>`
	`</MediaPropertiesLayouts>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

Note If the Finesse database is down or if there is a problem retrieving the media properties layout from the database, then a GET on https://<server>/finesse/api/MediaPropertiesLayouts (or on https://<server>/finesse/api/MediaPropertiesLayout/default) returns the system defined default media properties layout with an ID of 0.

MediaPropertiesLayout—Create

This API allows an administrator to create a custom media properties layout. Finesse supports up to 200 media properties layouts (1 default and 199 custom media properties layouts).

URI:	https://<FQDN>/finesse/api/MediaPropertiesLayout/
Example URI:	https://finesse1.xyz.com/finesse/api/MediaPropertiesLayout/
Security Constraints:	Only administrators can use this API.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<MediaPropertiesLayout>`
	`    <name>Layout name</name>`
	`    <description>Layout description</description>`
	`    <header>`
	`        <entry>`
	`            <displayName>Customer Name</displayName>`
	`            <mediaProperty>callVariable1</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`    </header>`
	`    <column>`
	`        <entry>`
	`            <displayName>Customer Name</displayName>`
	`            <mediaProperty>callVariable1</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`        <entry>`
	`            <displayName>Customer Acct#</displayName>`
	`            <mediaProperty>user.cisco.acctnum</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`    </column>`
	`    <column>`
	`        <entry>`
	`            <displayName>Support contract</displayName>`
	`            <mediaProperty>callVariable2</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`        <entry>`
	`            <displayName>Product calling about</displayName>`
	`            <mediaProperty>callVariable3</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`    </column>`
	`</MediaPropertiesLayout>`

Request Parameters:	name (required): Name of the media properties layout description (optional): Description of the media properties layout header (optional): Mapping for a single mediaProperty to be displayed with a label on the call details in the agent or supervisor desktop column (optional): Grouping of mediaProperties for agent or supervisor desktops entry (optional): Contains a displayName and mediaProperty combination displayName (required): Name of the field to be displayed to the agent or supervisor mediaProperty (required): Value of the entry to be displayed to the agent or supervisor matched with the displayName in the same entry showInPopOver: Indicates if the call variables to be displayed in the Call PopOver are based on the set value (true or false) uiEditable: Indicates if the call variable values can be edited in the agent and supervisor desktop (true or false)

HTTP Response:	200: Success **Note** Finesse successfully created the new media properties layout. The response contains an empty response body and a location header that denotes the absolute URL of the newly created MediaPropertiesLayout object. 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	   `<ApiError>`
	      `<ErrorType>Authorization Failure</ErrorType>`
	      `<ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	      `<ErrorData>jsmith</ErrorData>`
	   `</ApiError>`
	`</ApiErrors>`

MediaPropertiesLayout—Update

This API allows an administrator to update the media properties layout associated with the specified ID.

URI:	https://<FQDN>/finesse/api/MediaPropertiesLayout/{id}
Example URI:	https://finesse1.xyz.com/finesse/api/MediaPropertiesLayout/15
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<MediaPropertiesLayout>`
	`    <name>Layout name</name>`
	`    <description>Layout description</description>`
	`    <header>`
	`        <entry>`
	`            <displayName>Customer Name</displayName>`
	`            <mediaProperty>callVariable1</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`    </header>`
	`    <column>`
	`        <entry>`
	`            <displayName>Customer Name</displayName>`
	`            <mediaProperty>callVariable1</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`        <entry>`
	`            <displayName>Customer Acct#</displayName>`
	`            <mediaProperty>user.cisco.acctnum</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`    </column>`
	`    <column>`
	`        <entry>`
	`            <displayName>Support contract</displayName>`
	`            <mediaProperty>callVariable2</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`        <entry>`
	`            <displayName>Product calling about</displayName>`
	`            <mediaProperty>callVariable3</mediaProperty>`
	`            <showInPopOver>false</showInPopOver>`
	`            <uiEditable>false</uiEditable>`
	`        </entry>`
	`    </column>`
	`</MediaPropertiesLayout>`

Request Parameters:	name (required): Name of the media properties layout description (optional): Description of the media properties layout header (optional): Mapping for a single mediaProperty to be displayed with a label on the call details in the agent or supervisor desktop column (optional): Grouping of mediaProperties for agent or supervisor desktops entry (optional): Contains a displayName and mediaProperty combination displayName (required): Name of the field to be displayed to the agent or supervisor mediaProperty (required): Value of the entry to be displayed to the agent or supervisor matched with the displayName in the same entry showInPopOver: Indicates if the call variables to be displayed in the Call PopOver are based on the set value (true or false) uiEditable: Indicates if the call variable values can be edited in the agent and supervisor desktop (true or false)

HTTP Response:	200: Success 400: Parameter Missing 400: Invalid Input 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

MediaPropertiesLayout—Update Default Layout

This API allows an administrator to update the default media properties layout for the Finesse desktop.

Note Cisco Finesse supports this API for backward compatibility, but to update the default layout, developers must specify the default MediaPropertiesLayout ID in the MediaPropertiesLayout—Update API.

URI:	https://<FQDN>/finesse/api/MediaPropertiesLayout/default
Example URI:	https://finesse1.xyz.com/finesse/api/MediaPropertiesLayout/default
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	
	```
	<MediaPropertiesLayout>
	    <name>Default</name>
	    <description>default layout</description>
	    <header>
	        <entry>
	            <displayName>Customer Name</displayName>
	            <mediaProperty>callVariable1</mediaProperty>
	            <showInPopOver>false</showInPopOver>
	            <uiEditable>false</uiEditable>
	        </entry>
	    </header>
	    <column>
	        <entry>
	            <displayName>Customer Name</displayName>
	            <mediaProperty>callVariable1</mediaProperty>
	            <showInPopOver>false</showInPopOver>
	            <uiEditable>false</uiEditable>
	        </entry>
	        <entry>
	            <displayName>Customer Acct#</displayName>
	            <mediaProperty>user.cisco.acctnum</mediaProperty>
	            <showInPopOver>false</showInPopOver>
	            <uiEditable>false</uiEditable>
	        </entry>
	    </column>
	    <column>
	        <entry>
	            <displayName>Support contract</displayName>
	            <mediaProperty>callVariable2</mediaProperty>
	            <showInPopOver>false</showInPopOver>
	            <uiEditable>false</uiEditable>
	        </entry>
	        <entry>
	            <displayName>Product calling about</displayName>
	            <mediaProperty>callVariable3</mediaProperty>
	            <showInPopOver>false</showInPopOver>
	            <uiEditable>false</uiEditable>
	        </entry>
	    </column>
	</MediaPropertiesLayout>
	```

Request Parameters:	name (required): Name of the media properties layout description (optional): Description of the media properties layout header (optional): Contains displayName and mediaProperty that appears in the call header on the desktop column (optional): Grouping of media properties for the Finesse desktop (can contain a maximum of 10 entries) entry (optional): Contains a displayName and mediaProperty displayName (required): A label that describes the mediaProperty for that entry mediaProperty (required): The name of the variable for that entry showInPopOver: Indicates if the call variables to be displayed in the Call PopOver are based on the set value (true or false) uiEditable: Indicates if the call variable values can be edited in the agent and supervisor desktop (true or false)

HTTP Response:	200: Success 400: Invalid Input 400: Parameter Missing 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`     <ErrorData>The entry contained an invalid media property:`
	`      callVariable11</ErrorData>`
	`     <ErrorType>Invalid Input</ErrorType>`
	`     <ErrorMessage>HTTP Status code: 400 (Bad Request)`
	`        Api Error Type: Invalid Input`
	`        Error Message: Invalid media property name 'callVariable11'`
	`     </ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

MediaPropertiesLayout—Delete

This API allows an administrator to delete the custom media properties layout with the specified ID.

URI:	https://<FQDN>/finesse/api/MediaPropertiesLayout/{id}
Example URI:	https://finesse1.xyz.com/finesse/api/MediaPropertiesLayout/15
Security Constraints:	Only administrators can use this API. Administrators can only delete a media properties layout of type CUSTOM.
HTTP Method:	DELETE
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 401: Unauthorized 403: Forbidden 404: Not Found 500: Runtime exception

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

Note If you attempt to delete the default media properties layout, the system responds with one of the following errors, depending on the API you use for the operation:

API Used to Delete the Default Layout	HTTP Response	Details
https://<FQDN>/finesse/api/MediaPropertiesLayout/{id}	403 Forbidden	DELETE of the default media properties layout is forbidden with this API.
https://<FQDN>/finesse/api/MediaPropertiesLayout/default	405 Method Not Allowed	DELETE is not a supported operation with this API.

MediaPropertiesLayout API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The id maps to the primary key of the media properties layout entry.	—	
name	String	The name of the media properties layout.	—	Max length of 40 characters
description	String	The description of the media properties layout.	—	Max length of 128 characters
type	String	The type of media properties layout.	DEFAULT, CUSTOM	
header	Object	Contains a single entry (combination of displayName and mediaProperty) that appears in the call header on the desktop for each call.	—	

Parameter	Type	Description	Possible Values	Notes
column	Object	Grouping of media properties for agent and supervisor desktops. Contains a list of entry objects	—	Finesse supports up to two columns in the MediaProperties Layout object. Columns can contain up to 10 entries and can be empty. The first column supplied in a PUT is always the left column. The second column (if any) is always the right column.
-->entry	Object	A displayName and mediaProperty combination.	—	Each entry must contain one displayName and one mediaProperty. The displayName can be empty.
--->displayName	String	Part of an entry. A label that describes the mediaProperty for that entry (for example, Customer Name). The label appears on the Finesse desktop.	—	Maximum of 50 characters.

Parameter	Type	Description	Possible Values	Notes
--->mediaProperty	String	The name of the variable that is displayed on the Finesse desktop. Each entry contains exactly one mediaProperty.	Allowed strings include callVariable1 through callVariable10, any valid ECC variable (user.*), and the following Outbound Option variables: • BACampaign • BAAccountNumber • BAResponse • BAStatus • BADialedListID • BATimeZone • BABuddyName • BACustomerNumber (Unified CCX only)	Maximum of 32 characters.
--->showInPopOver	Boolean	Indicates the call variables to be displayed in the Call PopOver and in Supervisor team performance gadget based on the value.	TRUE, FALSE	Default value for this parameter is FALSE.
--->uiEditable	Boolean	Indicates the call variables that are editable in the agent and supervisor desktop. **Note** • Call variable (callVariable1 to callVariable10) values are editable. • ECC variable values are editable. • Amongst BA variables (campaign based outbound calls), only BA Response value is editable.	TRUE, FALSE	Default value for this parameter is FALSE.

MediaPropertiesLayout API Errors

Status	Error Type	Description
400	Bad Request	Request parameter is invalid.
400	Finesse API error	API error, such as: object is stale, violation of database constraint, and so on.
400	Invalid Input	At least one of the parameters provided is not valid.
400	Parameter Missing	At least one of the required parameters was not provided.
400	Maximum Exceeded	The maximum number of items has been exceeded.
400	Invalid Input	The user has selected more than five call variables when configuring call pop-over for a layout.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
401	Invalid Authorization User Specified	The authenticated user tried to use the identity that is not their own.
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server. The default media properties layout may not be deleted.
404	Not Found	Could not find the call variables layout with the specified ID.
405	Method Not Allowed	Unsupported operation is performed against an API. For example, if a DELETE or POST is attempted on: https://<FQDN>/finesse/api/MediaPropertiesLayout/default (which only supports GET and PUT).
500	Internal Server Error	Any runtime exception is caught and responded with this error.

PhoneBook

The PhoneBook object represents a phone book that contains contacts. Each PhoneBook object contains a Contacts summary object.

Phone books can be assigned globally (to all agents) or to specific teams. Finesse supports a maximum of 10 global phone books and 300 team phone books.

The PhoneBook object is structured as follows:

```
<PhoneBook>
    <uri>/finesse/api/PhoneBook/{id}</uri>
    <name></name>
    <type></type>
    <contacts>/finesse/api/PhoneBook/{id}/Contacts</contacts>
</PhoneBook>
```

PhoneBook APIs

PhoneBook—Get

This API allows an administrator to get a specific phone book.

URI:	https://<FQDN>/finesse/api/PhoneBook/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/34
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Finesse API Error 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<PhoneBook>`
	`   <uri>/finesse/api/PhoneBook/34</uri>`
	`   <name>Phonebook 1</name>`
	`   <type>GLOBAL</type>`
	`   <contacts>/finesse/api/PhoneBook/34/Contacts</contacts>`
	`</PhoneBook>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

PhoneBook—Get List

This API allows an administrator to get a list of all global and team phone books. Agents' personal phone books are not returned.

URI:	https://<FQDN>/finesse/api/PhoneBooks
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBooks
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 500: Internal Server Error

Example Response:	`<PhoneBooks>`
	`   <PhoneBook>`
	`      ...Full PhoneBook Object...`
	`   </PhoneBook>`
	`   <PhoneBook>`
	`      ...Full PhoneBook Object...`
	`   </PhoneBook>`
	`   <PhoneBook>`
	`      ...Full PhoneBook Object...`
	`   </PhoneBook>`
	`</PhoneBooks>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

PhoneBook—Create

This API allows an administrator to create a new phone book. The administrator specifies the name and type for the phone book.

URI:	https://<FQDN>/finesse/api/PhoneBook/
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/
Security Constraints:	Only administrators can use this API.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<PhoneBook>`
	`   <name>PhoneBook1</name>`
	`   <type>GLOBAL</type>`
	`</PhoneBook>`

Request Parameters:	name (required): The name of the phone book type (required): The type of phone book (GLOBAL or TEAM)
HTTP Response:	200: Success **Note** Finesse successfully created the new phone book. The server response contains an empty response body and a location header that denotes the absolute URL of the new phone book. 400: Invalid Input 400: Parameter Missing 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

PhoneBook—Update

This API allows an administrator to modify an existing phone book.

URI:	https://<FQDN>/finesse/api/PhoneBook/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/45
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<PhoneBook>`
	`   <name>PhoneBook2</name>`
	`   <type>TEAM</type>`
	`</PhoneBook>`

Request Parameters:	id (required): The database ID for the phone book name (required): The name of the phone book type (required): The type of phone book (GLOBAL or TEAM)
HTTP Response:	202: Successfully Accepted 400: In Use 400: Invalid Input 400: Parameter Missing 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

PhoneBook—Delete

This API allows an administrator to delete an existing phone book.

URI:	https://<FQDN>/finesse/api/PhoneBook/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/43
Security Constraints:	Only administrators can use this API.
HTTP Method:	DELETE

Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: In Use 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

PhoneBook—Import Contact List (CSV)

This API allows an administrator to replace all the contacts in a specific phonebook by importing a list of contacts in a comma-separated values (CSV) file. The CSV file can contain up to 6000 contacts. Cisco Finesse supports a system-wide total of 50,000 contacts.

All existing contacts in the phonebook are deleted before the new contacts are inserted. Contacts that contain errors are not inserted. Contacts that are error-free or contacts that contain missing or empty fields are inserted.

In general, the import is fault-tolerant. The CSV file is sent using standard web form syntax and is delivered to the Cisco Finesse server as multipart/form data.

This format is particular about formatting. Lines in the CSV file must be separated by carriage returns and newlines (\r\n). To import:

1. Use the PhoneBook—Get List API to get a list of all the global and team phonebooks. From the returned list, find the `id` of the phonebook containing the contacts that need to be replaced. The phonebook `id` can be found in the `uri` field.

2. Create a Web Form HTML file by copying the below HTML into a new file. In the `form action` field, replace <FQDN> with the FQDN of the Finesse server and <id> with the phonebook `id` obtained from Step 1. Save the file on your desktop as a HTML file. Example: `phonebook.html`.

```
<form action="https://<FQDN>/finesse/api/PhoneBook/<id>/Contacts/csvFileContent"
enctype="multipart/form-data" method="post">
     <p>
          File(s):
          <input type="file" name="datafile" size="40">
     </p>
     <div>
          <input type="submit" value="Import">
     </div>
</form>
```

3. Create a CSV file with the phonebook content you want to upload. Example: `pb.csv` (Also saved to the Desktop).

```
"First Name","Last Name","Phone Number","Notes"
"Agent","10001","20001","Sales"
"Agent","10002","20002","Service"
"Agent","10003","20011","Supervisor"
"","VVB","090011","HelloWorld"
"","Survivability","090011","To HelloWorld"
```

4. Run the `phonebook.html` file. A browser window opens.
5. Click **Browse** and select the `pb.csv` file.
6. Click **Import**.

URI:	https://<FQDN>/finesse/api/PhoneBook/<id>/Contacts/csvFileContent
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/34/Contacts/csvFileContent
Security Constraints:	Only administrators can use this API.
HTTP Method:	POST
Content Type:	text/CSV
Input/Output Format:	text/plain, text/CSV

Example HTML Form:	`<form action="https://finesse1.xyz.com/finesse/api/PhoneBook/34/Contacts/csvFileContent" enctype="multipart/form-data" method="post">`
	`    <p>`
	`        File(s):`
	`        <input type="file" name="datafile" size="40">`
	`    </p>`
	`    <div>`
	`        <input type="submit" value="Import">`
	`    </div>`
	`</form>`

HTTP Request:	`-----------------------------13290916118636`
	`Content-Disposition: form-data; name="phonebook"`
	`-----------------------------13290916118636`
	`Content-Disposition: form-data; name="datafile"; filename="pb.csv"`
	`Content-Type: application/vnd.ms-excel`
	
	`"First Name","Last Name","Phone Number","Notes"`
	`"Amanda","Cohen","6511234",""`
	`"Nicholas","Knight","6125551228","Sales"`
	`"Natalie","Lambert","9525559876","Benefits"`
	`"Joseph","Stonetree","6515557612","Manager"`

Request Parameters:	id (required): The database ID for the phonebook

HTTP Response:	202: Successfully Accepted **Note** This response indicates a successful completion of the request. The request is processed and the actual response is sent as part of and updated to the PhoneBook object. 400: Invalid Input 400: Maximum Exceeded 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

PhoneBook—Import Contact List (XML)

This API allows an administrator to replace all the contacts in a specific phone book by importing a collection of contacts. The collection can contain up to 6000 contacts.

All existing contacts in the phone book are deleted before the new contacts are inserted. Contacts that contain errors are not inserted.

URI:	https://<FQDN>/finesse/api/PhoneBook/<id>/Contacts
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/34/Contacts
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Contacts>`
	`   <Contact>`
	`      ...Full Contact Object...`
	`   </Contact>`
	`   <Contact>`
	`      ...Full Contact Object...`
	`   </Contact>`
	`   <Contact>`
	`      ...Full Contact Object`
	`   </Contact>`

Request Parameters:	id (required): The database ID for the phone book

HTTP Response:	202: Successfully Accepted **Note** This response indicates a successful completion of the request. The request is processed and the actual response is sent as part of and updated to the PhoneBook object. **Note** Some of the data could not be imported because it was invalid. The ErrorData field contains a list of lines that were not imported. This response indicates partial success because some data was uploaded. 400: Invalid Input 400: Maximum Exceeded 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

PhoneBook—Export Contact List

This API allows an administrator to export a list of contacts that belong to a specific phone book. The list is exported in CSV format.

URI:	https://<FQDN>/finesse/api/PhoneBook/<id>/Contacts/csvFileContent
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/34/Contacts/csvFileContent
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	text/CSV
Input/Output Format:	Multipart/form-data type=file

Example Exported CSV File:	`"First Name","Last Name","Phone Number","Notes"`
	`"Amanda","Cohen","6511234",""`
	`"Nicholas","Knight","6125551228","Sales"`
	`"Natalie","Lambert","9525559876","Benefits"`
	`"Joseph","Stonetree","6515557612","Manager"`

HTTP Response:	200: Success **Note** This response indicates a successful completion of the request. After a successful request, browser clients are prompted to save the returned content as a CSV file. 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

PhoneBook API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the PhoneBook object.	—	The id in the URI maps to the primary key of the phone book entry.
name	String	The name of the phone book.	—	
type	String	The type of phone book.	GLOBAL, TEAM	

PhoneBook API Errors

Status	Error Type	Description
400	Finesse API Error	API error such as the object is stale or does not exist.

Status	Error Type	Description
400	Invalid Input	One of the input parameters exceeded constraints. Contacts could not be imported because the data was invalid. The file may be empty or may not contain any valid lines. If the ErrorData field contains no lines, there may not be data to import. The multipart mime message may have been improperly formatted or did not contain a file. The multipart mime message may have been improperly formatted or did not contain a file. In this case, the existing records are overwritten.
400	In Use	The phone book is assiged to a team. You cannot change a team phone book to a global phone book if it is use. You cannot delete a phone book if it is use.
400	Maximum Exceeded	The maximum number of phone books or contacts has been exceeded.
400	Parameter Missing	A required parameter was not present in the request.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
401	Invalid Authorization User Specified	The authenticated user tried to use the identity of another user.
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
404	Not Found	The specified resource cannot be found.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

Contact

The Contact object represents a contact that can be assigned to a phone book. A phone book can contain up to 6000 contacts. Finesse supports a system-wide total of 50,000 contacts.

The Contact object is structured as follows:

```
<Contact>
   <firstName></firstName>
   <lastName></lastName>
   <phoneNumber></phoneNumber>
   <description></description>
   <uri>/finesse/api/PhoneBook/{phoneBookId}/Contact/{id}</uri>
</Contact>
```

Contact APIs

Contact—Get

This API allows an administrator to get a specific phone book contact.

URI:	https://<FQDN>/finesse/api/PhoneBook/<phoneBookId>/Contact/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/34/Contact/785
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<Contact>`
	`   <firstName>John</firstName>`
	`   <lastName>Doe</lastName>`
	`   <phoneNumber>5551234</phoneNumber>`
	`   <description>Accounts Manager</description>`
	`   <uri>/finesse/api/PhoneBook/34/Contact/785</uri>`
	`</Contact>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Authorization Failure</ErrorType>`
	`      <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`      <ErrorData>jsmith</ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

Contact—Get List

This API allows an administrator to get a list of contacts for a specific phone book.

URI:	https://<FQDN>/finesse/api/PhoneBook/<phoneBookId>/Contacts
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/34/Contacts
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<Contacts>`
	   `<Contact>`
	      `...Full Contact Object...`
	   `</Contact>`
	   `<Contact>`
	      `...Full Contact Object...`
	   `</Contact>`
	   `<Contact>`
	      `...Full Contact Object...`
	   `</Contact>`
	`</Contacts>`

Example Failure Response:	`<ApiErrors>`
	   `<ApiError>`
	      `<ErrorType>Authorization Failure</ErrorType>`
	      `<ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	      `<ErrorData>jsmith</ErrorData>`
	   `</ApiError>`
	`</ApiErrors>`

Contact—Create

This API allows an administrator to create a new phone book contact.

URI:	https://<FQDN>/finesse/api/PhoneBook/<phoneBookId>/Contact/

Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/34/Contact/
Security Constraints:	Only administrators can use this API.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Contact>`
	`   <firstName>Jerry</firstName>`
	`   <lastName>Green</lastName>`
	`   <phoneNumber>5554444</phoneNumber>`
	`   <description>Product Expert</description>`
	`</Contact>`

Request Parameters:	phoneBookId (required): Maps to the primary key of the phone book to which the contact belongs firstName (optional): The first name of the contact lastName (optional): The last name of the contact phoneNumber (required): The phone number of the contact description (optional): A description for the contact
HTTP Response:	200: Success **Note** Finesse successfully created the new contact. The server response contains an empty response body and a location header that denotes the absolute URL of the new contact. 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Contact—Update

This API allows an administrator to modify a specific phone book contact.

URI:	https://<FQDN>/finesse/api/PhoneBook/<phoneBookId>/Contact/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/45 /Contact/787

Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Contact>`
	`   <firstName>Marie</firstName>`
	`   <lastName>Brown</lastName>`
	`   <phoneNumber>5554444</phoneNumber>`
	`   <description>Product Expert</description>`
	`</Contact>`

Request Parameters:	phoneBookId (required): Maps to the primary key of the phone book to which the contact belongs id (required): Maps to the primary key of the contact entry firstName (optional): The first name of the contact lastName (optional): The last name of the contact phoneNumber (required): The phone number of the contact description (optional): A description for the contact
HTTP Response:	202: Successfully Accepted 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Contact—Delete

This API allows an administrator to delete an existing phone book contact.

URI:	https://<FQDN>/finesse/api/PhoneBook/<phoneBookId>/Contact/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/PhoneBook/43 /Contact/1523
Security Constraints:	Only administrators can use this API.
HTTP Method:	DELETE
Content Type:	Application/XML

Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorType>Authorization Failure</ErrorType>`
	`          <ErrorMessage>UNAUTHORIZED</ErrorMessage>`
	`          <ErrorData>jsmith</ErrorData>`
	`     </ApiError>`
	`</ApiErrors>`

Contact API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the Contact object.	—	The phoneBookId in the URI maps to the primary key of the phone book to which the contact belongs. The id in the URI maps to the primary key of the contact entry.
firstName	String	The first name of the contact.	—	Maximum of 128 characters.
lastName	String	The last name of the contact.	—	Maximum of 128 characters.
phoneNumber	String	The phone number for the contact.	—	Maximum of 32 characters.
description	String	A description of the contact.	—	Maximum of 128 characters.

Contact API Errors

Status	Error Type	Description
400	Bad Request	The request body is invalid.
400	Finesse API Error	API error such as the object is stale or does not exist.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
404	Not Found	The specified resource cannot be found.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

Workflow

The Workflow object represents a workflow that can be assigned to a team. Workflows manage agent activity based on call events. Workflows have triggers and conditions, which are used to determine whether the associated actions are run. The Workflow object contains the following subobjects: TriggerSet, ConditionSet, and workflowActions. The Workflow object is structured as follows:

```
<Workflow>
   <uri>/finesse/api/Workflow/{id}</uri>
   <name></name>
   <description></description>
   <media></media>
   <TriggerSet>
      <type></type>
      <name></name>
      <allowOverlappingCallWorkflow></allowOverlappingCallWorkflow>
      <triggers>
         <Trigger>
            <Variable>
               <name></name>
               <node></node>
               <type></type>
            </Variable>
            <comparator></comparator>
            <value></value>
         </Trigger>
         <Trigger>
            <Variable>
               <name></name>
               <node></node>
               <type></type>
            </Variable>
            <comparator></comparator>
            <value></value>
         </Trigger>
      </triggers>
   </TriggerSet>
   <ConditionSet>
      <applyMethod></applyMethod>
      <conditions>
         <Condition>
            <Variable>
               <name></name>
               <type></type>
            </Variable>
            <comparator></comparator>
            <value></value>
         </Condition>
         <Condition>
            <Variable>
               <name></name>
               <type></type>
            </Variable>
            <comparator></comparator>
            <value></value>
         </Condition>
      </conditions>
   </ConditionSet>
   <workflowActions>
      <WorkflowAction>
         <name></name>
         <type></type>
         <uri>/finesse/api/WorkflowAction/{id}</uri>
      </WorkflowAction>
      <WorkflowAction>
         <name></name>
         <type></type>
         <uri>/finesse/api/WorkflowAction/{id}</uri>
      </WorkflowAction>
   </workflowActions>
</Workflow>
```

The following SYSTEM TriggerSets are defined by the Finesse system. When you create a workflow, you need only specify the name and type of SYSTEM. The TriggerSets are automatically expanded when retrieved by the User—Get list of workflows API.

CALL_ARRIVES

```
<TriggerSet>
    <type>SYSTEM</type>
    <name>CALL_ARRIVES</name>
    <triggers>
        <Trigger>
            <Variable>
                <name>mediaType</name>
                <node>//Dialog/mediaType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_EQUAL</comparator>
            <value>Voice</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>callType</name>
                <node>//Dialog/mediaProperties/callType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>ACD_IN,PREROUTE_ACD_IN,PREROUTE_DIRECT_AGENT,TRANSFER,OVERFLOW_IN,
            OTHER_IN,AGENT_OUT,OUT,OUTBOUND,OUTBOUND_CALLBACK,OUTBOUND_PERSONAL_CALLBACK,
        AGENT_INSIDE,OFFERED,CONSULT,CONSULT_OFFERED,CONSULT_CONFERENCE,CONFERENCE,
        TASK_ROUTED_BY_ICM,TASK_ROUTED_BY_APPLICATION,VOICE_CALL_BACK,NON_ACD,
        SUPERVISOR_BARGE_IN,NULL</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>state</name>
                <node>//Dialog/participants/Participant/mediaAddress
                [.='${extension}']/../state</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>ALERTING,ACTIVE,HELD</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>fromAddress</name>
                <node>//Dialog/fromAddress</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_NOT_EQUAL</comparator>
            <value>${extension}</value>
        </Trigger>
    </triggers>
</TriggerSet>
```

CALL_ANSWERED

```
<TriggerSet>
    <type>SYSTEM</type>
    <name>CALL_ANSWERED</name>
    <triggers>
        <Trigger>
            <Variable>
                <name>mediaType</name>
                <node>//Dialog/mediaType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_EQUAL</comparator>
            <value>Voice</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>callType</name>
                <node>//Dialog/mediaProperties/callType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>ACD_IN,PREROUTE_ACD_IN,PREROUTE_DIRECT_AGENT,TRANSFER,OVERFLOW_IN,
            OTHER_IN,AGENT_OUT,OUT,OUTBOUND,OUTBOUND_CALLBACK,OUTBOUND_PERSONAL_CALLBACK,
            AGENT_INSIDE,OFFERED,CONSULT,CONSULT_OFFERED,CONSULT_CONFERENCE,CONFERENCE,
            TASK_ROUTED_BY_ICM,TASK_ROUTED_BY_APPLICATION,VOICE_CALL_BACK,NON_ACD,
            SUPERVISOR_BARGE_IN,NULL</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>state</name>
                <node>//Dialog/participants/Participant/mediaAddress
                [.='${extension}']/../state</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_EQUAL</comparator>
            <value>ACTIVE</value>
        </Trigger>
    </triggers>
</TriggerSet>
```

CALL_ENDS

```
<TriggerSet>
    <type>SYSTEM</type>
    <name>CALL_ENDS</name>
    <triggers>
        <Trigger>
            <Variable>
                <name>mediaType</name>
                <node>//Dialog/mediaType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_EQUAL</comparator>
            <value>Voice</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>callType</name>
                <node>//Dialog/mediaProperties/callType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>ACD_IN,PREROUTE_ACD_IN,PREROUTE_DIRECT_AGENT,TRANSFER,OVERFLOW_IN,
        OTHER_IN,AGENT_OUT,OUT,OUTBOUND,OUTBOUND_CALLBACK,OUTBOUND_PERSONAL_CALLBACK,
        AGENT_INSIDE,OFFERED,CONSULT,CONSULT_OFFERED,CONSULT_CONFERENCE,CONFERENCE,
        TASK_ROUTED_BY_ICM,TASK_ROUTED_BY_APPLICATION,VOICE_CALL_BACK,NON_ACD,
        SUPERVISOR_BARGE_IN,NULL</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>state</name>
                <node>//Dialog/participants/Participant/mediaAddress
                [.='${extension}']/../state</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>DROPPED,WRAP_UP</value>
        </Trigger>
    </triggers>
</TriggerSet>
```

CALL_IS_MADE

```
<TriggerSet>
    <type>SYSTEM</type>
    <name>CALL_IS_MADE</name>
    <triggers>
        <Trigger>
            <Variable>
                <name>mediaType</name>
                <node>//Dialog/mediaType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_EQUAL</comparator>
            <value>Voice</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>callType</name>
                <node>//Dialog/mediaProperties/callType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>ACD_IN,PREROUTE_ACD_IN,PREROUTE_DIRECT_AGENT,TRANSFER,OVERFLOW_IN,
        OTHER_IN,AGENT_OUT,OUT,OUTBOUND,OUTBOUND_CALLBACK,OUTBOUND_PERSONAL_CALLBACK,
        AGENT_INSIDE,OFFERED,CONSULT,CONSULT_OFFERED,CONSULT_CONFERENCE,CONFERENCE,
        TASK_ROUTED_BY_ICM,TASK_ROUTED_BY_APPLICATION,VOICE_CALL_BACK,NON_ACD,
        SUPERVISOR_BARGE_IN,NULL</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>state</name>
                <node>//Dialog/participants/Participant/mediaAddress
                [.='${extension}']/../state</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>ALERTING,INITIATED,FAILED,ACTIVE,HELD</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>fromAddress</name>
                <node>//Dialog/fromAddress</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_EQUAL</comparator>
            <value>${extension}</value>
        </Trigger>
    </triggers>
</TriggerSet>
```

CALL_IS_PREVIEWED

```
<TriggerSet>
    <type>SYSTEM</type>
    <name>CALL_IS_PREVIEWED</name>
    <triggers>
        <Trigger>
            <Variable>
                <name>mediaType</name>
                <node>//Dialog/mediaType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_EQUAL</comparator>
            <value>Voice</value>
        </Trigger>
        <Trigger>
            <Variable>
                <name>callType</name>
                <node>//Dialog/mediaProperties/callType</node>
                <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>OUTBOUND_PREVIEW,OUTBOUND_CALLBACK_PREVIEW,OUTBOUND_DIRECT_PREVIEW,
            OUTBOUND_PERSONAL_CALLBACK_PREVIEW</value>
        </Trigger>
    </triggers>
    <allowOverlappingCallWorkflow>true</allowOverlappingCallWorkflow>
</TriggerSet>
```

Workflow APIs

Workflow—Get

This API allows an administrator to get a specific Workflow object.

URI:	https://<FQDN>/finesse/api/Workflow/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/Workflow/195
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	

```
<Workflow>
   <uri>/finesse/api/Workflow/195</uri>
   <name>Workflow A</name>
   <description>Workflow description</description>
   <media>Media Channel</media>
   <TriggerSet>
      <type>SYSTEM</type>
      <name>CALL_ARRIVES</name>
      <triggers>
         <Trigger>
            <Variable>
               <name>mediaType</name>
               <node>//Dialog/mediaType</node>
               <type>CUSTOM</type>
            </Variable>
            <comparator>IS_EQUAL</comparator>
            <value>Voice</value>
         </Trigger>
         <Trigger>
            <Variable>
               <name>callType</name>
               <node>//Dialog/mediaProperties/callType</node>
               <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>ACD_IN,PREROUTE_ACD_IN,PREROUTE_
             DIRECT_AGENT,TRANSFER,OVERFLOW_IN,
             OTHER_IN,AGENT_OUT,OUT,OUTBOUND,OUTBOUND_
             CALLBACK,OUTBOUND_PERSONAL_CALLBACK,AGENT_INSIDE,
             OFFERED,CONSULT,CONSULT_OFFERED,CONSULT_CONFERENCE,
             CONFERENCE,TASK_ROUTED_BY_ICM,TASK_ROUTED_BY_
             APPLICATION,VOICE_CALL_BACK,NON_ACD,SUPERVISOR_
             BARGE_IN,NULL</value>
         </Trigger>
         <Trigger>
            <Variable>
               <name>state</name>

<node>//Dialog/participants/Participant/mediaAddress[.=${userExtension}]/../state</node>

               <type>CUSTOM</type>
            </Variable>
            <comparator>IS_IN_LIST</comparator>
            <value>ALERTING,ACTIVE,HELD</value>
         </Trigger>
      </triggers>
   </TriggerSet>
   <ConditionSet>
      <applyMethod>ALL</applyMethod>
      <conditions>
         <Condition>
            <Variable>
               <name>callVariable1</name>
               <type>SYSTEM</type>
            </Variable>
            <comparator>CONTAINS</comparator>
            <value>1234</value>
         </Condition>
         <Condition>
            <Variable>
               <name>user.foo.bar[1}</name>

<node>/dialogs/Dialog/mediaProperties/callvariables/CallVariable/name[.="user.foo.bar[1]"]/../value</node>
```

```
                  <type>CUSTOM</type>
               </Variable>
               <comparator>IS_NOT_EMPTY</comparator>
            </Condition>
         </conditions>
      </ConditionSet>
      <workflowActions>
         <WorkflowAction>
            <name>Google</name>
            <type>BROWSER_POP</type>
            <uri>/finesse/api/WorkflowAction/1234</uri>
         </WorkflowAction>
         <WorkflowAction>
            <name>Company Web Page</name>
            <type>BROWSER_POP</type>
            <uri>/finesse/api/WorkflowAction/9876</uri>
         </WorkflowAction>
      </workflowActions>
</Workflow>
```

Example Failure Response:

```
<ApiErrors>
   <ApiError>
      <ErrorData>Workflow 10009 not found.</ErrorData>
      <ErrorType>Not Found</ErrorType>
      <ErrorMessage>HTTP Status code:404 (Not Found)
         Api Error Type:  Not Found
         Error Message: Workflow not found with an id of 10009
      </ErrorMessage>
   </ApiError>
</ApiErrors>
```

Workflow—Get List

This API allows an administrator to get a list of workflows.

URI:	https://<FQDN>/finesse/api/Workflows
Example URI:	https://finesse1.xyz.com/finesse/api/Workflows
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Response:	`<Workflows>`
	`   <Workflow>`
	`      ...Full Workflow Object...`
	`   </Workflow>`
	`   <Workflow>`
	`      ...Full Workflow Object...`
	`   </Workflow>`
	`   <Workflow>`
	`      ...Full Workflow Object...`
	`   </Workflow>`
	`</Workflows>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorData>Database read/write error</ErrorData>`
	`      <ErrorType>Bad Request</ErrorType>`
	`      <ErrorMessage>`
	`         HTTP Status code: 400 (Bad Request)`
	`         Api Error Type:  Bad Request`
	`         Error Message: Database read/write error`
	`      </ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Workflow—Create

This API allows an administrator to create a new workflow. Finesse supports a maximum of 100 workflows.

Note If you provide two or more duplicate tags during a POST, the value of the last duplicate tag is processed and all other duplicate tags are ignored.

URI:	https://<FQDN>/finesse/api/Workflow/
Example URI:	https://finesse1.xyz.com/finesse/api/Workflow/
Security Constraints:	Only administrators can use this API.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Workflow>`
	`   ...Full Workflow Object...`
	`</Workflow>`

Request Parameters:	id (required): Maps to the primary key of the workflow entry name (required): The name of the workflow description (optional): A description of the workflow Media (optional): The media of the workflow TriggerSet (required): A set of events that cause the conditions to be evaluated ConditionSet (optional): A set of conditions that determine if the workflow is run workflowActions (optional): A list of workflow actions to run if the trigger and conditions are satisfied
HTTP Response:	200: Success **Note** Finesse successfully created the new workflow. The server response contains an empty response body and a location header that denotes the absolute URL of the new phone book. 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorData>Duplicate Workflow name.</ErrorData>`
	`      <ErrorType>Database constraint violation</ErrorType>`
	`      <ErrorMessage>`
	`         HTTP Status code: 400 (Bad Request)`
	`         Api Error Type:  Database constraint violation`
	`         Error Message: A workflow with the same name`
	`         already exists`
	`      </ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Workflow—Update

This API allows an administrator to update an existing workflow.

If the attributes (name, description, TriggerSet, ConditionSet, workflowActions) for the specified workflow do not change, the request does not need to include those attributes. If an attribute is not specified, the current value is retained. However, you must specify at least one attribute in the request.

If you only want to change the description of the workflow, you can make the following request:

```
<Workflow>
   <description>New description</description>
</Workflow>
```

Note If you provide two or more duplicate tags during a PUT, the value of the last duplicate tag is processed and all other duplicate tags are ignored.

URI:	https://<FQDN>/finesse/api/Workflow/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/Workflow/769
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Workflow>`
	 `   ...Workflow Object...`
	 `</Workflow>`

Request Parameters:	id (required): Maps to the primary key of the workflow entry name (optional): The name of the workflow description (optional): A description of the workflow Media (optional): The media of the workflow TriggerSet (optional): A set of events that cause the conditions to be evaluated ConditionSet (optional): A set of conditions that determine if the workflow is run workflowActions (optional): A list of workflow actions to run if the trigger and conditions are satisfied
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorData>For update, at least one field must be set.</ErrorData>`
	
	`      <ErrorType>Invalid Input</ErrorType>`
	`      <ErrorMessage>`
	`         HTTP Status code: 400 (Bad Request)`
	`         Api Error Type:  Invalid Input`
	`         Error Message: Updating a Workflow requires specifying at`
	`         least one value to be changed.`
	`      </ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Workflow—Delete

This API allows an administrator to delete an existing workflow. The administrator references the existing Workflow object by its ID.

URI:	https://<FQDN>/finesse/api/Workflow/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/Workflow/768
Security Constraints:	Only administrators can use this API.
HTTP Method:	DELETE
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorData>Workflow 1009 not found.</ErrorData>`
	`      <ErrorType>Not Found</ErrorType>`
	`      <ErrorMessage>`
	`         HTTP Status code: 404 (Not Found)`
	`         Api Error Type:  Not Found`
	`         Error Message: Workflow not found with an id of 1009`
	`      </ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Workflow API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the Workflow object.	—	The id in the URI maps to the primary key of the workflow.
name	String	The name of the workflow.	—	Must be unique. Maximum of 40 characters.
description	String	A description of the workflow.	—	Maximum of 128 characters.
media	String	Media channel of the workflow	—	Media channel maps to the media id. **Note** Domain List can be obtained from the MediaDomain API. • For Unified CCE, you can define custom media channels for Voice and Digital Channels. • For Unified CCX, the media channels are: • Voice with media id as 1. • Chat with media id as Chat. • Email with media id as Email. **Note** If no media channels are specified, Voice is set as the default media.

Parameter	Type	Description	Possible Values	Notes
TriggerSet	Object	A set of events that cause the conditions to be evaluated.	—	
ConditionSet	Object	A set of conditions that determine whether the workflow runs.	—	You can assign up to five conditions to a workflow.
workflowActions	Object	A list of workflow actions to run if the trigger and its conditions are met. Actions run in the order in which they appear in this list.	—	You can assign up to five workflow actions to a workflow. When getting a workflow or list of workflows, this list contains summary workflow actions (name, type, and URL). When creating or updating a workflow, only the URI is required in each workflow action. For more information, see WorkflowAction, on page 331.

ConditionSet Parameters

Parameter	Type	Description	Possible Values	Notes
applyMethod	String	Determines whether any or all of the conditions must be met for the workflow to run.	ANY, ALL	
conditions	Object	A list of conditions for the workflow.	—	Maximum of five conditions for a workflow. A workflow with no conditions is specified by a conditions parameter with no Condition elements.
Condition	Object	Information about a workflow condition.	—	

Parameter	Type	Description	Possible Values	Notes
Variable	Object	A piece of data from the Trigger event used to filter the event.	—	Leading and trailing spaces are removed from the variable during evaluation. Comma-separated values in a list also have leading and trailing spaces removed. If the value contains only spaces, it is treated as an empty value.
comparator	String	The operator used to compare the event variable to the desired value.	IS_EQUAL, IS_NOT_EQUAL, BEGINS_WITH, ENDS_WITH, CONTAINS, IS_EMPTY, IS_NOT_EMPTY, IS_IN_LIST, IS_NOT_IN_LIST	
value	String	The value to compare the event variable with.	When type is SYSTEM, valid values are CALL_ARRIVES, CALL_ANSWERED, CALL_ENDS, CALL_IS_MADE, and CALL_IS_PREVIEWED.	If the comparator is IS_IN_LIST or IS_NOT_IN_LIST, the value is one of a comma-separated list of values. If an explicit comma is needed, it must be escaped with a backslash (\,). If a backslash is needed, it must be escaped with a backslash (\\) (for example, apple,slash\\ here,comma\,here,ball).

TriggerSet Parameters

Parameter	Type	Description	Possible Values	Notes
type	String	The type of TriggerSet.	SYSTEM	

Parameter	Type	Description	Possible Values	Notes
name	String	The name of the TriggerSet	When type is SYSTEM, valid values are CALL_ARRIVES, CALL_ANSWERED, CALL_ENDS, CALL_IS_MADE, and CALL_IS_PREVIEWED.	
allow Overlapping CallWorkflow	Boolean	Indicates whether workflow for a second simultaneous call can fir while the call for this trigger is in process.	TRUE, FALSE	Default for this parameter is FALSE.
triggers	Object	List of Trigger subobjects.	—	For workflow admin, this field is not returned and is ignored if the type is SYSTEM.

Trigger Parameters

Parameter	Type	Description	Possible Values	Notes
Variable	Object	A piece of data from the trigger event to be used to filter the event. Contains a name, node, and type.	—	
name	String	A unique name for the variable. Used as a readable, unique key for the variable.	—	
node	String	The XPath to use to extract the value of the variable from an XMPP event that might contain it.	—	

Parameter	Type	Description	Possible Values	Notes
type	String	Indicates whether this is a system or custom variable.	SYSTEM, CUSTOM	SYSTEM variables are name references to the values returned by SystemVariable and do not require a node value. CUSTOM variables are self-defining and require a node and a unique name that does not conflict with any system variable.

Nodes can contain the following predefined variables as part of their XPath. When the node is evaluated, the current value as received in the most recent User event will be substituted in place of the variable. Variables are surrounded by ${} when specified in XPath as shown in the table below.

Note These variables are a subset of those defined by the SystemVariable resource

SYSTEM variables are name references to the values returned by SystemVariable and do not require a node value. CUSTOM variables are self-defining and require a node and a unique name that does not conflict with any system variable.

Variable Name	Value	Data Type
${userExtension}	The extension this user is currently using.	String
${userLoginId}	The login ID of the user.	String
${userLoginName}	The user's login name.	String
${userTeamName}	The name of the team the user belongs to.	String
${userTeamId}	The ID of the team the user belongs to.	String
${userFirstName}	The first name of the user.	String
${userLastName}	The last name of the user.	String

Workflow API Errors

Status	Error Type	Description
400	Bad Request	The request body is invalid.
400	Finesse API Error	API error such as the object is stale or does not exist.

Status	Error Type	Description
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
404	Not Found	The specified resource cannot be found.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

WorkflowAction

The WorkflowAction object represents a workflow action that can be assigned to a workflow. Finesse supports a system-wide maximum of 100 workflow actions.

The WorkflowAction object is structured as follows:

```
<WorkflowAction>
    <uri>/finesse/api/WorkflowAction/{id}</uri>
    <name></name>
    <type></type>
    <handledBy></handledBy>
    <params>
        <Param>
            <name><name>
            <value></value>
        </Param>
        <Param>
            <name></name>
            <value></value>
        </Param>
    </params>
    <actionVariables>
        <ActionVariable>
            <name></name>
            <type></type>
        </ActionVariable>
    </actionVariables>
</WorkflowAction>
```

There are two types of workflow actions: BROWSER_POP and HTTP_REQUEST.

The BROWSER_POP type is structured as follows:

```
<WorkflowAction>
    <uri>/finesse/api/WorkflowAction/{id}</uri>
    <name>DuckDuckGo</name>
    <type>BROWSER_POP</type>
    <handledBy>FINESSE_DESKTOP</handledBy>
    <params>
```

```
        <Param>
     <name>path<name>
           <value>http://www.example.com?q=${callVariable1}</value>
     </Param>
     <Param>
     <name>windowName</name>
           <value>theWindow</value>
      </Param>
    </params>
    <actionVariables>
     <ActionVariable>
     <name>callVariable1</name>
           <type>SYSTEM</type>
     </ActionVariable>
    </actionVariables>
</WorkflowAction>
```

The HTTP_REQUEST type is structured as follows:

```
<WorkflowAction>
    <name>Test with Content Type</name>
    <type>HTTP_REQUEST</type>
    <handledBy>FINESSE_DESKTOP</handledBy>
        <Param>
        <name>path</name>
           <value>http://www.example.com?q=${callVariable1}</value>
        </Param>
        <Param>
            <name>method</name>
            <value>PUT</value>
        </Param>
        <Param>
            <name>authenticationType</name>
            <value>BASIC</value>
        </Param>
        <Param>
            <name>location</name>
            <value>OTHER</value>
        </Param>
        <Param>
            <name>contentType</name>
            <value>application/xml</value>
        </Param>
        <Param>
            <name>body</name>
            <value>${callVariable1},${callVariable2}</value>
        </Param>
    </params>
    <actionVariables>
        <ActionVariable>
            name>callVariable1</name>
            <type>SYSTEM</type>
        </ActionVariable>
        <ActionVariable>
            <name>callVariable2</name>
            <type>SYSTEM</type>
        </ActionVariable>
    </actionVariables>
</WorkflowAction>
```

WorkflowAction APIs

WorkflowAction—Get

This API allows an administrator to get a specific WorkflowAction object.

URI:	https://<FQDN>/finesse/api/WorkflowAction/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/WorkflowAction/674
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<WorkflowAction>`
	`   ...Full WorkflowAction Object...`
	`</WorkflowAction>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorData>Action 674 not found.</ErrorData>`
	`      <ErrorType>Not Found</ErrorType>`
	`      <ErrorMessage>HTTP Status code:404 (Not Found)`
	`         Api Error Type:  Not Found`
	`         Error Message: Workflow not found with an id of 674`
	`      </ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

WorkflowAction—Get List

This API allows an administrator to get a list of workflow actions.

URI:	https://<FQDN>/finesse/api/WorkflowActions
Example URI:	https://finesse1.xyz.com/finesse/api/WorkflowActions

Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Response:	`<WorkflowActions>`
	   `<WorkflowAction>`
	      `<name>WorkflowAction 1</name>`
	         `<type>HTTP</name>`
	         `<uri>/finesse/api/WorkflowAction/{id}</uri>`
	   `</WorkflowAction>`
	   `<WorkflowAction>`
	      `<name>WorkflowAction 2</name>`
	         `<type>DELAY</name>`
	         `<uri>/finesse/api/WorkflowAction/{id}</uri>`
	   `</WorkflowAction>`
	`</WorkflowActions>`

Example Failure Response:	`<ApiErrors>`
	   `<ApiError>`
	      `<ErrorData>Database read/write error</ErrorData>`
	      `<ErrorType>Bad Request</ErrorType>`
	      `<ErrorMessage>`
	         `HTTP Status code: 400 (Bad Request)`
	         `Api Error Type:  Bad Request`
	         `Error Message: Database read/write error`
	      `</ErrorMessage>`
	   `</ApiError>`
	`</ApiErrors>`

WorkflowAction—Create

This API allows an administrator to create a new workflow action.

Note If you provide two or more duplicate tags during a POST, the value of the last duplicate tag is processed and all other duplicate tags are ignored.

URI:	https://<FQDN>/finesse/api/WorkflowAction/

Example URI:	https://finesse1.xyz.com/finesse/api/WorkflowAction/
Security Constraints:	Only administrators can use this API.
HTTP Method:	POST
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<WorkflowAction>`
	`   ...Full WorkflowAction Object...`
	`</WorkflowAction>`

Request Parameters (Browser Pop):	name (required): The name of the workflow action type (required): The type of workflow action handledBy (required): Indicates what handles the action params (required): List of Params for the workflow action actionVariables (required): list of actionVariables for the workflow path (required): The path to use in the action windowName (optional): The window name to pop open
Request Parameters (HTTP Request):	name (required): The name of the workflow action type (required): The type of workflow action handledBy (required): Indicates what handles the action params (required): List of Params for the workflow action actionVariables (required): list of actionVariables for the workflow path (required): The path to use in the action method (required): The method to use in the request authenticationType (optional): The authentication type to use in the request location (required): Whether the request is to Finesse or a third party contentType (optional): The value of the content type header to send with the request body (optional): The body to send with the request

HTTP Response:	200: Success **Note** Finesse successfully created the new workflow action. The server response contains an empty response body and a location header that denotes the absolute URL of the new workflow action. 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorData>Action Type is invalid.</ErrorData>`
	`      <ErrorType>Invalid Input</ErrorType>`
	`      <ErrorMessage>`
	`         HTTP Status code: 400 (Bad Request)`
	`         Api Error Type:  Invalid Input`
	`         Error Message: type is invalid`
	`      </ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

WorkflowAction—Update

This API allows an administrator to update an existing workflow action.

If the attributes (name, description, TriggerSet, ConditionSet, workflowActions) for the specified workflow do not change, the request does not need to include those attributes. If an attribute is not specified, the current value is retained. However, you must specify at least one attribute in the request.

If you only want to change the description of the workflow, you can make the following request:

```
<Workflow>
   <description>New description</description>
</Workflow>
```

Note If you provide two or more duplicate tags during a PUT, the value of the last duplicate tag is processed and all other duplicate tags are ignored.

URI:	https://<FQDN>/finesse/api/WorkflowAction/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/WorkflowAction/769
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<WorkflowAction>`
	`   ...WorkflowAction Object...`
	`</WorkflowAction>`

Request Parameters:	id (required): Maps to the primary key of the workflowAction entry name (required): The name of the workflow action type (required): The type of workflow action handledBy (required): Indicates what handles the action params (required): List of Params for the workflow action actionVariables (required): list of actionVariables for the workflow
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorData>Duplicate Action name.</ErrorData>`
	`      <ErrorType>Database constraint violation</ErrorType>`
	`      <ErrorMessage>`
	`         HTTP Status code: 400 (Bad Request)`
	`         Api Error Type:  Database constraint violation`
	`         Error Message: An action with the same name already`
	`         exists`
	`      </ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

WorkflowAction—Delete

This API allows an administrator to delete an existing workflow action. The administrator references the existing WorkflowAction object by its ID.

URI:	https://<FQDN>/finesse/api/WorkflowAction/<id>
Example URI:	https://finesse1.xyz.com/finesse/api/WorkflowAction/768
Security Constraints:	Only administrators can use this API.
HTTP Method:	DELETE
Content Type:	Application/XML
Input/Output Format:	XML
HTTP Request:	—

HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorData>Action 768 not found.</ErrorData>`
	`      <ErrorType>Not Found</ErrorType>`
	`      <ErrorMessage>`
	`         HTTP Status code: 404 (Not Found)`
	`         Api Error Type:  Not Found`
	`         Error Message: This is not a valid action`
	`      </ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

WorkflowAction API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the WorkflowAction object.	—	The id in the URI maps to the primary key of the WorkflowAction.
name	String	The name of the workflow action.	—	Must be unique. Maximum of 64characters.
type	String	The type of workflow action	BROWSER_POP, HTTP_REQUEST	

Parameter	Type	Description	Possible Values	Notes
handledBy	String	Indicates what handles the action when it is triggered by a workflow.	FINESSE_DESKTOP, OTHER	For FINESSE_DESKTOP, the Finesse workflow engine runs the action. For OTHER, the action event is published on the OpenAJAX hub but is not run by the Finesse desktop. This allows a third-party gadget to run the action.
params	Object	A list of Param subobjects.	—	
-->Param	Object	Includes a name and value pair.	—	Params are flexible and can contain any value. Validation is based on the type of the WorkflowAction in which they are contained. See the following tables for more information.
--->name	String	The name of the parameter.	—	
--->value	String	The value of the parameter.	—	
actionVariables	Object	List of ActionVariable subobjects.	—	
-->ActionVariable	Object	Set of information about one ActionVariable.	—	You can assign up to five ActionVariable parameters to a workflow.
--->name	String	The name of the variable.	—	Maximum of 32 characters.

Parameter	Type	Description	Possible Values	Notes
--->node	String	The XPath to extract from the dialog XML.	—	Maximum of 500 characters. SYSTEM variables are name references to the values returned by SystemVariable and do not require a node value. CUSTOM variables are self-defining and require a node and a unique name that does not conflict with any system variable.
--->type	String	Indicates the type of variable	CUSTOM, SYSTEM	
--->testValue	String	The value used to test the variable.	—	Maximum of 128 characters.

Param Values (BROWSER_POP)

Parameter	Description	Possible Values	Size	Required?
path	The path to use in the BROWSER_POP action	The URL path is validated only to make sure its length is at least 1 and no longer than the maximum length. It is up to the user to provide a valid URL. Variables can be embedded into the URL by using a dollar sign and curly braces. For example: `http://www.example.com?q=${callVariable1}` causes the workflow engine to substitute the value of callVariable1 into the path. If a literal curly brace or dollar sign is needed in the URL, it must be escaped with a backslash (for example, \{). A literal backslash must be escaped with another backslash (\\).	500	Yes
windowName	The window name to pop open	The window name is passed to the browser Window Open method by the work flow engine. The value can be any string other than _parent, _self, or _top. It can also be an empty string or missing entirely, in which case the workflow engine passes _blank to the Window Open method.	40	No

Param (HTTP_REQUEST)

Parameter	Description	Possible Values	Size	Required?

path	The path to use in the HTTP_REQUEST action	The URL path is validated only to make sure its length is at least 1 and no longer than the maximum length. It is up to the user to provide a valid URL. Variables can be embedded into the URL by using a dollar sign and curly braces. For example:
		`http://www.example.com?q=${callVariable1}`
		will cause the workflow engine to substitute the value of callVariable1 into the path. If a literal curly brace or dollar sign is needed in the URL, they must be escaped with a backslash (e.g. \{). A literal backslash must be escaped with another backslash (e.g. \\).
		When location is FINESSE, the protocol, host, and port should not be specified. These will be inferred automatically by Finesse when it runs the REST request. For example, to send a dialog request for dialog id 32458, the following URL should be entered:
		`/finesse/api/Dialog/32458`
			500	Yes

method	The method to use in the HTTP_REQUEST	PUT, POST		Yes
authenticationType	The authentication type to use in the HTTP_REQUEST	BASIC: A basic access authentication header is included in the REST request each time it is made. NONE: No authentication is used with the request, no authentication headers or other negotiation is done as part of the request.		No
location	Defines if the HTTP_REQUEST is to Finesse or to a third party application	FINESSE: The request is made to Finesse and passes the credentials of the currently logged-in user NONE: No credentials are included as part of the request.		No
contentType	The value of the content type header to send with the HTTP_REQUEST	The content type is only validated to ensure it does not exceed the maximum length. You must make sure you provide a valid content type. If the parameter is empty, no content type header is sent with the HTTP_REQUEST.	500	No

body	The body to send with the HTTP_REQUEST	A free form text string that is included in the body of the request. It may be JSON, XPATH or any other format. It is not validated. If xml is included in the value it must be well formed xml. Variables may be embedded into the body by using a dollar sign curly braces. For example:
		`<foo>${callVariable1}</foo>`
		causes the workflow engine to substitute the value of callVariable1 into the body. If a literal curly brace or dollar sign is needed in the body it must be escaped with a backslash:
		`\{`
		A literal backslash must be escaped with another backslash :
		`\\`
			2000	No

WorkflowAction API Errors

Status	Error Type	Description
400	Bad Request	The request body is invalid.
400	Finesse API Error	API error such as the object is stale or does not exist.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
404	Not Found	The specified resource cannot be found.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

Team

The Team object represents a team and the resources associated with that team. For more information, see Team, on page 159.

The administrator uses the Team configuration APIs to assign or unassign resources (such as reason codes, wrap-up reasons, phonebooks, layout configuration, and workflows) to a specific team.

Team APIs

Team—Get List

This API allows an administrator to get a list of teams. The team must have agents or supervisors assigned to it for the team to appear in the retrieved list.

URI:	https://<FQDN>/finesse/api/Teams
Example URI:	https://finesse1.xyz.com/finesse/api/Teams
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Response:	`<Teams>`
	`   <Team>`
	`      ...Summary Team Object...`
	`   </Team>`
	`   <Team>`
	`      ...Summary Team Object...`
	`   </Team>`
	`   <Team>`
	`      ...Summary Team Object...`
	`   </Team>`
	`</Teams>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Unauthorized</ErrorType>`
	`      <ErrorMessage>The user is not authorized to`
	`       perform this operation.</ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Team—Get List of Reason Codes

This API allows an administrator to get a list of reason codes for the specified category assigned to a specific team. The list is in the same format as defined in the section *ReasonCode*.

URI:	https://<FQDN>/finesse/api/Team/<id>/ReasonCodes?category=<category>
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/ReasonCodes?category=NOT_READY
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<ReasonCodes category="NOT_READY">`
	`      <ReasonCode>`
	`           ... Full Reason Code Object ...`
	`      </ReasonCode>`
	`      <ReasonCode>`
	`           ... Full Reason Code Object ...`
	`      </ReasonCode>`
	`      <ReasonCode>`
	`           ... Full Reason Code Object ...`
	`      </ReasonCode>`
	`....`
	`</ReasonCodes>`

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorData>500</ErrorData>`
	`          <ErrorType>finesse.api.team.team_assignment_invalid_`
	`           team&</ErrorType>`
	`          <ErrorMessage>HTTP Status code: 404 (Not Found)`
	`           Api Error Type:finesse.api.team.team_assignment_invalid_team`
	
	`           Error Message:`
	`           This is not a valid team</ErrorMessage>`
	`     </ApiError>`
	`</ApiErrors>`

Team—Update List of Reason Codes

This API allows an administrator to assign or unassign a list of reason codes of the specified category to a team.

If multiple users try to update the reason code for the same team at the same time, the changes made by the last user to update overwrite the changes made by the other users.

This list includes all reason codes of the specified category that are assigned to a team. Any reason codes that you assign or unassign overwrite the current reason code list.

Note The category attribute of the ReasonCodes tag is not required for the update. If it is included in the request, it is ignored. However, all the reason codes in the list must have a category specified in the category query parameter. Inclusion of a reason code whose category does not match results in a Finesse API error (Status 400).

URI:	https://<FQDN>/finesse/api/Team/<Id>/ReasonCodes?category=<category>
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/ReasonCodes?category=NOT_READY
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<ReasonCodes>`
	`      <ReasonCode>`
	`           <uri>/finesse/api/ReasonCode/123</uri>`
	`      </ReasonCode>`
	`      <ReasonCode>`
	`           <uri>/finesse/api/ReasonCode/456</uri>`
	`      </ReasonCode>`
	`      <ReasonCode>`
	`           <uri>/finesse/api/ReasonCode/789</uri>`
	`      </ReasonCode>`
	`....`
	`</ReasonCodes>`

Request Parameters:	id (required): The database ID for the team category (required): The category of reason code (NOT_READY or LOGOUT)

HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorData>category NOT_READ is invalid</ErrorData>`
	`          <ErrorType>Invalid Input</ErrorType>`
	`          <ErrorMessage>HTTP Status code:400 (Bad Request)`
	`           Api Error Type:Invalid Input`
	`           Error Message:Category must be NOT_READY`
	`           or LOGOUT</ErrorMessage>`
	`     </ApiError>`
	`</ApiErrors>`

Team—Get List of Wrap-Up Reasons

This API allows an administrator to get a list of wrap-up reasons assigned to a specific team. The list is in the same format as defined in the section WrapUpReason, on page 267.

URI:	https://<FQDN>/finesse/api/Team/<id>/WrapUpReasons
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/WrapUpReasons
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	
	```
	<WrapUpReasons>
	      <WrapUpReason>
	           ... Full WrapUpReason Object ...
	      </WrapUpReason>
	      <WrapUpReason>
	           ... Full WrapUpReason Object ...
	      </WrapUpReason>
	      <WrapUpReason>
	           ... Full WrapUpReason Object ...
	      </WrapUpReason>
	....
	</WrapUpReasons>
	```

Example Failure Response:	
	```
	<ApiErrors>
	     <ApiError>
	          <ErrorData>500</ErrorData>
	          <ErrorType>finesse.api.team.team_assignment_invalid_
	           team&</ErrorType>
	          <ErrorMessage>HTTP Status code: 404 (Not Found)
	           Api Error Type:finesse.api.team.team_assignment_
	           invalid_team
	           Error Message:
	           This is not a valid team</ErrorMessage>
	     </ApiError>
	</ApiErrors>
	```

Team—Update List of Wrap-Up Reasons

This API allows an administrator to assign or unassign a list of wrap-up reasons to a team.

This API restricts the maximum number of non-global wrap-up reasons that can be assigned to a single team to 100.

If multiple users try to update the wrap-up reasons for the same team at the same time, the changes made by the last user to update overwrite the changes made by the other users.

This list includes all wrap-up reasons that are assigned to a team. Any wrap-up reasons that you assign or unassign overwrite the current wrap-up reason list.

URI:	https://<FQDN>/finesse/api/Team/<Id>/WrapUpReasons
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/WrapUpReasons
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<WrapUpReasons>`
	`      <WrapUpReason>`
	`           <uri>/finesse/api/WrapUpReason/123</uri>`
	`      </WrapUpReason>`
	`      <WrapUpReason>`
	`           <uri>/finesse/api/WrapUpReason/456</uri>`
	`      </WrapUpReason>`
	`      <WrapUpReason>`
	`           <uri>/finesse/api/WrapUpReason/789</uri>`
	`      </WrapUpReason>`
	`....`
	`</WrapUpReasons>`

Request Parameters:	id (required): The database ID for the team
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 400: Maximum Exceeded 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`     <ErrorData>574</ErrorData>`
	`     <ErrorType>finesse.api.team.team_assignment_`
	`      invalid_team</ErrorType>`
	`     <ErrorMessage>HTTP Status code: 404 (Not Found)`
	`      Api Error Type:finesse.api.team.team_assignment_`
	`      invalid_team Error Message:`
	`      This is not a valid team</ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Team—Get List of Phone Books

This API allows an administrator to get a list of phone books assigned to a specific team. The list is in the same format as defined in the section PhoneBook, on page 297.

URI:	https://<FQDN>/finesse/api/Team/<id>/PhoneBooks
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/PhoneBooks
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—

Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<PhoneBooks>`
	`      <PhoneBook>`
	`           ... Full PhoneBook Object ...`
	`      </PhoneBook>`
	`      <PhoneBook>`
	`           ... Full PhoneBook Object ...`
	`      </PhoneBook>`
	`      <PhoneBook>`
	`           ... Full PhoneBook Object ...`
	`      </PhoneBook>`
	`....`
	`</PhoneBooks>`

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorData>574</ErrorData>`
	`          <ErrorType>finesse.api.team.team_assignment_invalid_`
	`           team&</ErrorType>`
	`          <ErrorMessage>HTTP Status code: 404 (Not Found)`
	`           Api Error Type:finesse.api.team.team_assignment_`
	`           invalid_team`
	`           Error Message:`
	`           This is not a valid team</ErrorMessage>`
	`     </ApiError>`
	`</ApiErrors>`

Team—Update List of Phone Books

This API allows an administrator to assign or unassign a list of phone books to a team.

If multiple users try to update the phone books for the same team at the same time, the changes made by the last user to update overwrite the changes made by the other users.

This list includes all phone books that are assigned to a team. Any phone books that you assign or unassign overwrite the current phone book list.

URI:	https://<FQDN>/finesse/api/Team/<Id>/PhoneBooks
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/PhoneBooks
Security Constraints:	Only administrators can use this API.

HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<PhoneBooks>`
	`      <PhoneBook>`
	`           <uri>/finesse/api/PhoneBook/123</uri>`
	`      </PhoneBook>`
	`      <PhoneBook>`
	`           <uri>/finesse/api/PhoneBook/456</uri>`
	`      </PhoneBook>`
	`      <PhoneBook>`
	`           <uri>/finesse/api/PhoneBook/789</uri>`
	`      </PhoneBook>`
	`....`
	`</PhoneBooks>`

Request Parameters:	id (required): The database ID for the team
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`     <ErrorData>574</ErrorData>`
	`     <ErrorType>finesse.api.team.team_assignment_`
	`      invalid_team</ErrorType>`
	`     <ErrorMessage>HTTP Status code: 404 (Not Found)`
	`      Api Error Type:finesse.api.team.team_assignment_`
	`      invalid_team Error Message:`
	`      This is not a valid team</ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Team—Get Layout Configuration

This API allows an administrator to get the layout configuration assigned to a specific team.

URI:	https://<FQDN>/finesse/api/Team/<id>/LayoutConfig
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/LayoutConfig
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET

Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<TeamLayoutConfig>`
	`   <useDefault>false</useDefault>`
	`   <layoutxml>`
	`      <finesseLayout xmlns="http://www.cisco.com/vtg/finesse">`
	`         <layout>`
	`            <role>Agent</role>`
	`            ...`
	`         </layout>`
	`         <layout>`
	`     <role>Supervisor</role>`
	`     ...`
	`      </layout>`
	`    </finesseLayout>`
	`  </layoutxml>`
	`</TeamLayoutConfig>`

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorData>574</ErrorData>`
	`          <ErrorType>finesse.api.team.team_assignment_invalid_`
	`           team&</ErrorType>`
	`          <ErrorMessage>HTTP Status code: 404 (Not Found)`
	`           Api Error Type:finesse.api.team.team_assignment_`
	`           invalid_team`
	`           Error Message:`
	`           This is not a valid team</ErrorMessage>`
	`     </ApiError>`
	`</ApiErrors>`

Team—Update Layout Configuration

This API allows an administrator to assign or unassign a layout configuration to a team.

If multiple users try to update the layout configuration for the same team at the same time, the changes made by the last user to update overwrite the changes made by the other users.

URI:	https://<FQDN>/finesse/api/Team/<Id>/LayoutConfig
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/LayoutConfig

Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	Example of assigning a team-specific layout:
	`<TeamLayoutConfig>`
	`   <useDefault>false</useDefault>`
	`   <layoutxml>`
	`      <finesseLayout xmlns="http://www.cisco.com/vtg/finesse">`
	`         <layout>`
	`            <role>Agent</role>`
	`            ...`
	`         </layout>`
	`         <layout>`
	`     <role>Supervisor</role>`
	`     ...`
	`      </layout>`
	`    </finesseLayout>`
	`  </layoutxml>`
	`</TeamLayoutConfig>`
	Example of assigning the default layout to a team:
	`<TeamLayoutConfig>`
	`   <useDefault>true</useDefault>`
	`</TeamLayoutConfig>`

Request Parameters:	id (required): The database ID for the team useDefault (required): Whether to use the default desktop layout for this team layoutxml (required if useDefault is false): The XML data that determines the layout of the Finesse desktop
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	
	```
	<ApiErrors>
	   <ApiError>
	     <ErrorData>574</ErrorData>
	     <ErrorType>finesse.api.team.team_assignment_
	      invalid_team</ErrorType>
	     <ErrorMessage>HTTP Status code: 404 (Not Found)
	      Api Error Type:finesse.api.team.team_assignment_
	      invalid_team Error Message:
	      This is not a valid team</ErrorMessage>
	   </ApiError>
	</ApiErrors>
	```

Team—Get List of Workflows

This API allows an administrator to get a list of workflows assigned to a specific team. The list is in the same format as defined in the section Workflow, on page 313.

URI:	https://<FQDN>/finesse/api/Team/<id>/Workflows
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/Workflows
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	
	```
	<Workflows>
	      <Workflow>
	           ... Summary Workflow Object ...
	      </Workflow>
	      <Workflow>
	           ... Summary Workflow Object ...
	      </Workflow>
	      <Workflow>
	           ... Summary Workflow Object ...
	      </Workflow>
	....
	</Workflows>
	```

Example Failure Response:	`<ApiErrors>`
	`     <ApiError>`
	`          <ErrorData>574</ErrorData>`
	`          <ErrorType>finesse.api.team.team_assignment_invalid_`
	`           team&</ErrorType>`
	`          <ErrorMessage>HTTP Status code: 404 (Not Found)`
	`           Api Error Type:finesse.api.team.team_assignment_`
	`           invalid_team`
	`           Error Message:`
	`           This is not a valid team</ErrorMessage>`
	`     </ApiError>`
	`</ApiErrors>`

Team—Update List of Workflows

This API allows an administrator to assign or unassign a list of workflows to a team.

If multiple users try to update the workflows for the same team at the same time, the changes made by the last user to update overwrite the changes made by the other users.

This list includes all workflows that are assigned to a team. Any workflows that you assign or unassign overwrite the current workflow list.

Note Because the order in which workflows are evaluated is important, the order of the workflows in the list is preserved in the GET method (see Team—Get List of Workflows, on page 353).

URI:	https://<FQDN>/finesse/api/Team/<Id>/workflows
Example URI:	https://finesse1.xyz.com/finesse/api/Team/574/Workflows
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<Workflows>`
	`      <Workflow>`
	`           <uri>/finesse/api/Workflow/123</uri>`
	`      </Workflow>`
	`      <Workflow>`
	`           <uri>/finesse/api/Workflow/456</uri>`
	`      </Workflow>`
	`      <Workflow>`
	`           <uri>/finesse/api/Workflow/789</uri>`
	`      </Workflow>`
	`....`
	`</Workflows>`

Request Parameters:	id (required): The database ID for the team

HTTP Response:	200: Success 400: Bad Request 400: Finesse API Error 401: Authorization Failure 401: Invalid Authorization User Specified 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`     <ErrorData>574</ErrorData>`
	`     <ErrorType>finesse.api.team.team_assignment_`
	`      invalid_team</ErrorType>`
	`     <ErrorMessage>HTTP Status code: 404 (Not Found)`
	`      Api Error Type:finesse.api.team.team_assignment_`
	`      invalid_team Error Message:`
	`      This is not a valid team</ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

Team API Parameters

Parameter	Type	Description	Possible Values	Notes
uri	String	The URI to get a new copy of the Team, ReasonCode, WrapUpReason, LayoutConfig, or Workflow object.	—	
id	String	The unique identifier for the team.		
name	String	The name of the team.	—	
category	String	Specifies the type of reason code.	NOT_READY, LOGOUT	
useDefault	Boolean	Determines whether to use the default desktop layout for this team.	true, false	
layoutxml	String	The XML data that determines the desktop layout.	—	If useDefault is set to true and the layoutxml is provided in a request, the layoutxml is ignored.

Team API Errors

Status	Error Type	Description
400	Bad Request	The request body is invalid.
400	Finesse API Error	API error such as the object is stale or does not exist.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
404	Not Found	The specified resource cannot be found.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

SystemVariable

The SystemVariable object represents a variable that can be extracted from a Finesse event object and displayed on the Finesse desktop or used in a workflow.

The SystemVariable object is structured as follows:

```
<SystemVariable>
   <name></name>
   <node></node>
</SystemVariable>
```

SystemVariable APIs

SystemVariable—List

This API allows an administrator to get a list of all system variables.

Note The Outbound variable BACustomerNumber only appears in the response when Finesse is deployed with Unified CCX.

URI:	https://<FQDN>/finesse/api/SystemVariables
Example URI:	https://finesse1.xyz.com/finesse/api/SystemVariables

Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 403: Forbidden 500: Internal Server Error

Example Response:	

```
<SystemVariables>
   <SystemVariable>
      <name>callVariable1</name>
      <node>>//Dialog/mediaProperties/callvariables/CallVariable/
       name[.="callVariable1"]/../value</node>
   </SystemVariable>
   <SystemVariable>
      <name>callVariable2</name>
      <node>//Dialog/mediaProperties/callvariables/CallVariable/
       name[.="callVariable2"]/../value</node>
   </SystemVariable>
   <SystemVariable>
      <name>callVariable3</name>
      <node>//Dialog/mediaProperties/callvariables/CallVariable/
       name[.="callVariable3"]/../value</node>
   </SystemVariable>
   ...Other callVariables (4 through 10)...
   <SystemVariable>
      <name>BAAccountNumber</name>
      <node>//Dialog/mediaProperties/callvariables/CallVariable/
       name[.="callVariable3"]/../value</node>
   </SystemVariable>
   <SystemVariable>
      <name>callVariable5</name>
      <node>//Dialog/mediaProperties/callvariables/CallVariable/
       name[.="BAAccountNumber"]/../value</node>
   </SystemVariable>
   <SystemVariable>
      <name>BABuddyName</name>
      <node>//Dialog/mediaProperties/callvariables/CallVariable/
       name[.="BABuddyName"]/../value</node>
   </SystemVariable>
   ...Other Outbound Variables...
   <SystemVariable>
      <name>DNIS</name>
      <node>//Dialog/mediaProperties/DNIS</node>
   <SystemVariable>
      <name>fromAddress</name>
      <node>//Dialog/fromAddress</node>
   </SystemVariable>
   <SystemVariable>
      <name>Extension</name>
      <node>//User/Extension</node>
   </SystemVariable>
   <SystemVariable>
      <name>loginId</name>
      <node>//User/loginId</node>
   </SystemVariable>
   <SystemVariable>
      <name>teamName</name>
      <node>//User/teamName</node>
   </SystemVariable>
   <SystemVariable>
      <name>teamId</name>
      <node>//User/teamId</node>
   </SystemVariable>
   <SystemVariable>
      <name>firstName</name>
      <node>//User/firstName</node>
   </SystemVariable>
   <SystemVariable>
      <name>lastName</name>
      <node>//User/lastName</node>
   </SystemVariable>
```

	`</SystemVariables>`
Example Failure Response:	No API errors are returned. Responses are 401/403/404 Errors.

SystemVariable API Parameters

Parameter	Type	Description	Possible Values	Notes
name	String	A unique name for the variable.	—	The name is used as a readable, unique key for the variable. Maximum of 32 characters.
node	String	The XPath to use to extract the value of this variable from an XMPP event that may contain the variable.	—	Maximum of 500 characters.

SystemVariable API Errors

Status	Error Type	Description
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session). The user is not authorized to use the API (the user is not an administrator).
403	Forbidden	The user attempted to run the API against the secondary Finesse server. Configuration APIs cannot be run against the secondary Finesse server.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

CHAPTER 5

Cisco Finesse Serviceability APIs

SystemInfo

The SystemInfo object represents the Finesse system and provides high-level configuration and state information such as the deployment type, the current system state, hostnames of the finesse nodes, and other details.

The SystemInfo object is structured as follows:

```
<SystemInfo>
    <currentTimestamp></currentTimestamp>
    <deploymentType></deploymentType>
    <lastCTIHeartbeatStatus></lastCTIHeartbeatStatus>
    <lastSuccessCTIHeartbeatTime></lastSuccessCTIHeartbeatTime>
    <ctiVersion></ctiVersion>
    <ctiHeartbeatInterval></ctiHeartbeatInterval>
    <ctiTimeInMMode></ctiTimeInMMode>
    <ctiMMode></ctiMMode>
    <finesseTimeInMMode></finesseTimeInMMode>
    <finesseMMode></finesseMMode>
    <license></license>
    <peripheralId></peripheralId>
    <primaryNode>
        <host></host>
    </primaryNode>
    <secondaryNode>
        <host></host>
    </secondaryNode>
    <status></status>
    <statusReason></statusReason>
    <systemAuthMode></systemAuthMode>
    <timezoneOffset></timezoneOffset>
    <uri></uri>
    <xmppDomain></xmppDomain>
    <xmppPubSubDomain></xmppPubSubDomain>
    <ctiServers>
        <ctiServer>
```

```
            <host></host>
            <connectedDuration></connectedDuration>
            <active></active>
        </ctiServer>
        <ctiServer>
            <host></host>
            <connectedDuration></connectedDuration>
            <active></active>
        </ctiServer>
    </ctiServers>
</SystemInfo>
```

SystemInfo APIs

SystemInfo—Get

This API allows a user to get information about the Finesse system.

URI:	https://<FQDN>/finesse/api/SystemInfo
Example URI:	https://finesse1.xyz.com/finesse/api/SystemInfo
Security Constraints:	**Access via proxy:** If this API is accessed via proxy, agents, supervisors, or administrators credentials are required. **Note** For a non-authenticated API via proxy, refer to the **Desktop Configuration** section in the *Cisco Finesse Desktop Interface API Guide*. **Access via non-proxy:** If this API is not accessed through a proxy, credentials are not required. However, when the webservice property `secureSystemInfo` is true, credentials are required. By default the value of `secureSystemInfo` is false.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 500: Internal Server Error

Example Response:

```
<SystemInfo>
    <currentTimestamp>2020-05-05T10:47:57.691Z</currentTimestamp>
    <deploymentType>UCCE</deploymentType>

<lastCTIHeartbeatStatus>success</lastCTIHeartbeatStatus><lastSuccessCTIHeartbeatTime>1588675675720</lastSuccessCTIHeartbeatTime>

    <ctiVersion>24</ctiVersion>
    <ctiHeartbeatInterval>2</ctiHeartbeatInterval>
    <ctiTimeInMMode>-1</ctiTimeInMMode>
    <ctiMMode></ctiMMode>
    <finesseTimeInMMode>32</finesseTimeInMMode>
    <finesseMMode>IN_PROGRESS</finesseMMode>
    <license></license>
    <peripheralId>5001</peripheralId>
    <primaryNode>
        <host>finesse25.autobot.cvp</host>
    </primaryNode>
    <secondaryNode>
        <host>finesse125.autobot.cvp</host>
    </secondaryNode>
    <status>IN_SERVICE</status>
    <statusReason/>
    <systemAuthMode>NON_SSO</systemAuthMode>
    <timezoneOffset>-420</timezoneOffset>
    <uri>/finesse/api/SystemInfo</uri>
    <xmppDomain>finesse25.autobot.cvp</xmppDomain>
    <xmppPubSubDomain>pubsub.finesse25.autobot.cvp</xmppPubSubDomain>
    <ctiServers>
        <ctiServer>
            <host>pga.cisco.com</host>
            <connectedDuration>10</connectedDuration>
            <active>true</active>
        </ctiServer>
        <ctiServer>
            <host>pgb.cisco.com</host>
            <connectedDuration>-1</connectedDuration>
            <active>false</active>
        </ctiServer>
    </ctiServers>
</SystemInfo>
```

Example Failure Response:

```
<ApiErrors>
   <ApiError>
      <ErrorType>Internal Server Error</ErrorType>
      <ErrorMessage>Runtime Exception</ErrorMessage>
      <ErrorData></ErrorData>
   </ApiError>
</ApiErrors>
```

SystemInfo API Parameters

Parameter	Type	Description	Possible Values	Notes
currentTimeStamp	String	The current time (GMT time) in the following format: YYYY-MM-DDThh:MM:ss.SSZ	—	—

Parameter	Type	Description	Possible Values	Notes
deploymentType	String	The type of deployment for Cisco Finesse.	UCCE, UCCX	—
lastCTIHeartbeatStatus	String	The heartbeat request from Finesse server to CTI server.	success, failure	Once the heartbeat request is sent, Cisco Finesse waits for the heartbeat confirmation.
lastSuccessCTIHeartbeatTime	Integer	The last successful heartbeat time between the Cisco Finesse server and the CTI server.	—	—
ctiVersion	Integer	The CTI protocol version with which the Cisco Finesse server is connected to the CTI server.	—	—
ctiHeartbeatInterval	Integer	The heartbeat interval between the Cisco Finesse server and the CTI server in seconds.	—	—
ctiTimeInMMode	Integer	The total time (in seconds) that the CTI server is in maintenance mode. The time will be negative if the CTI server to which the Finesse server is connected to is not in maintenance mode.	—	This parameter is applicable only for Unified CCE deployment versions 12.6(1) or later.

Parameter	Type	Description	Possible Values	Notes
ctiMMode	String	Indicates the status of the CTI server in maintenance mode.	INITIATING: Finesse server accepts the CTI maintenance mode request and waits for the CTI response. IN_PROGRESS: Maintenance mode request is received and accepted by the CTI server. Finesse server is now going to connect to the standby CTI server.	This parameter is applicable only for Unified CCE deployment versions 12.6(1) or later.
finesseTimeInMMode	Integer	The total time (in seconds) that the Cisco Finesse server is in maintenance mode. The time will be negative if the Cisco Finesse server is not in maintenance mode.	—	—

Parameter	Type	Description	Possible Values	Notes
finesseMMode	String	Indicates the status of the Cisco Finesse server in maintenance mode.	If there is no value in this field, then maintenance mode is not initiated on this system. IN_PROGRESS: Maintenance mode request is received and accepted by the Cisco Finesse server. Cisco Finesse is waiting for all the agents to be connected to the other side. COMPLETED: All the agents have successfully signed in to the other side. Cisco Finesse shuts down the services. FAILED: The maintenance mode did not complete successfully.	—
license	String	The Unified CCX license.	STANDARD, ENHANCED, or PREMIUM	This parameter is blank for Unified CCE deployments.
peripheralId	String	The ID of the Unified CCE peripheral to which Cisco Finesse is connected.	—	This parameter is blank for Unified CCX deployments.
primaryNode - host	String	The hostname or IP address.	—	Available for both the nodes.

Parameter	Type	Description	Possible Values	Notes
status	String	The state of the Cisco Finesse system.	IN_SERVICE: The system is in service and usual operations are accepted. OUT_OF_SERVICE: The system is out of service and usual operations result in a 503 Service Unavailable response.	—
statusReason	String	The reason for which Cisco Finesse system is down.	Possible out-of-service scenarios returned by Cisco Finesse system: • Cisco Finesse Database is down. • Cisco Finesse Notification Service is down. • Cisco Finesse connection to CTI Server is down. • CTI Peripheral ID xxx is down. • System is initializing. • Local Unified CCX Engine is not in Service. (Unified CCX only)	This parameter is blank when Finesse system is IN_SERVICE.
systemAuthMode	String	Information about the system authentication mode.	SSO or non-SSO	Hybrid is for Unified CCE deployment.

Parameter	Type	Description	Possible Values	Notes
timezoneOffset	Integer	The difference (in minutes) between the server time and GMT time.	—	For example, a value of 300 means the server time is GMT + 5 hours. A value of -300 means the server time is GMT - 5 hours.
uri	String	The URI to get a new copy of the SystemInfo object.	—	—
xmppDomain	String	The XMPP server domain.	—	—
xmppPubSubDomain	String	The XMPP server pubsub domain.	—	—
ctiServers	Collection	The list of configured CTI servers for Finesse.	—	—
-->ctiServer	Object	Information about a configured CTI server.	—	—
-->host	String	The hostname of the CTI server.	—	—
-->connectedDuration	Integer	The total time (in seconds) that the Cisco Finesse server has been connected to this particular CTI server. The time will be negative if the Cisco Finesse server is not currently connected to this CTI server.	—	—

Parameter	Type	Description	Possible Values	Notes
active	Boolean	Indicates whether the Cisco Finesse is connected to the active CTI server.	true, false	• Unified CCE deployments supporting CTI protocol version 24 and above - Agent PG supports parallel connections to both the PG's, with one in Active mode and another in Passive mode. • Unified CCX and Unified CCE deployments with CTI protocol version prior to 24—Indicates the CTI server on which the Cisco Finesse is connected.

SystemInfo API Errors

Status	Error Type	Description
500	Internal Server Error	Any runtime exception is caught and responded with this error.

Finesse MaintenanceMode

The Finesse MaintenanceMode object represents the state of the Finesse maintenance mode.

The Finesse MaintenanceMode object is structured as follows:

```
<MaintenanceMode>
    <status></status>
    <statusReason></statusReason>
    <usersLoggedIn>
        <desktop></desktop>
        <fippa></fippa>
        <thirdPartyClients></thirdPartyClients>
    </usersLoggedIn>
    <pendingUserDisconnections></pendingUserDisconnections>
    <estimatedCompletionTime></estimatedCompletionTime>
</MaintenanceMode>
```

Finesse MaintenanceMode APIs

Finesse MaintenanceMode—Get

This API allows the user to retrieve the current information about the Finesse maintenance mode.

URI:	https://<FQDN>/finesse/api/MaintenanceMode
Example URI:	https://finesse1.xyz.com/finesse/api/MaintenanceMode
Security Constraints:	Administrators, agents, and supervisors can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Response:	`<MaintenanceMode>`
	`    <status>IN_PROGRESS</status>`
	`    <usersLoggedIn>`
	`        <desktop>385</desktop>`
	`        <fippa>10</fippa>`
	`        <thirdPartyClients>5</thirdPartyClients>`
	`    </usersLoggedIn>`
	`    <pendingUserDisconnections>400</pendingUserDisconnections>`
	`    <estimatedCompletionTime>300</estimatedCompletionTime>`
	`</MaintenanceMode>`

Example Failure Response:	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Internal Server Error</ErrorType>`
	`      <ErrorMessage>Runtime Exception</ErrorMessage>`
	`      <ErrorData></ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`

Finesse MaintenanceMode—Update

This API allows the user to update the existing information about the Finesse maintenance mode.

URI:	https://<FQDN>/finesse/api/MaintenanceMode
Example URI:	https://finesse1.xyz.com/finesse/api/MaintenanceMode
Security Constraints:	Only administrators can use this API.
HTTP Method:	PUT
Content Type:	Application/XML
Input/Output Format:	XML

HTTP Request:	`<MaintenanceMode>`
	 `    <status>IN_PROGRESS</status>`
	 `</MaintenanceMode>`

Request Parameters	status(required) : The new state of the maintenance mode the administrator wants to be in (only IN_PROGRESS is valid).
HTTP Response:	200: Success 400: Bad Request, Invalid Input 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Example Failure Response:	`<ApiErrors>`
	 `   <ApiError>`
	 `      <ErrorType>Internal Server Error</ErrorType>`
	 `      <ErrorMessage>Runtime Exception</ErrorMessage>`
	 `      <ErrorData></ErrorData>`
	 `   </ApiError>`
	 `</ApiErrors>`

Finesse MaintenanceMode API Parameters

Parameter	Type	Description	Possible Values	Notes
status	String	The state of the Finesse maintenance mode request.	IN_PROGRESS: Maintenance request is received and accepted by the Cisco Finesse server. Cisco Finesse is waiting for all the agents to be connected to the alternate node. COMPLETED: All the agents have successfully logged in to the alternate node. Cisco Finesse has shut down the services. FAILED: The agent login to the alternate node is not completed or the maintenance mode is not completed within the configured time.	—
usersLoggedIn	—	The information about the users logged in at the time when the maintenance mode started.	—	—
-->desktop	Integer	The total number of Finesse desktop users logged in.	—	—
-->fippa	Integer	The total number of Finesse IP Phone Agent (IPPA) application users logged in.	—	—
-->thirdpartyclients	Integer	The total number of users logged in from a third-party client.	—	—

Parameter	Type	Description	Possible Values	Notes
pendingUserDisconnections	Integer	The number of agents that are connected to the server in maintenance mode and scheduled to move to the alternate node.	—	—
estimatedCompletionTime	Integer	The estimated time (in seconds) required to complete the maintenance operation.	—	—

Finesse MaintenanceMode API Errors

Status	Error Type	Description
400	Bad Request	The request is malformed or incomplete.
400	Invalid Input	One of the parameters provided as part of the user input is invalid or not recognized.
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session).
401	Invalid Authorization User Specified	The authenticated user tried to make a request for another user.
404	Not Found	The resource specified is invalid or does not exist.
500	Internal Server Error	Any runtime exception is caught and responded with this error (for example, when connection is not established with CTI server or any other component).

ConnectedUsersInfo

The ConnectedUsersInfo object retrieves the real-time list of connected users. Administrators can use this information to find the number of users and the type of connectivity to the Finesse server to support maintenance or other administrative activities. This information fetched is not instantaneous and is refreshed every five seconds.

The ConnectedUsersInfo object is structured as follows:

```
<ConnectedUsersInfo>
  <userSummary>
    <desktopUsers></desktopUsers>
    <fippaUsers></fippaUsers>
    <thirdPartyUsers></thirdPartyUsers>
    <totalConnectedUsers></totalConnectedUsers>
  </userSummary>
  <uri>/finesse/api/ConnectedUsersInfo</uri>
</ConnectedUsersInfo>
```

ConnectedUsersInfo APIs

ConnectedUsersInfo—Summary

This API provides the summary of the connected users information.

URI:	https://<FQDN>/finesse/api/ConnectedUsersInfo
Example URI:	https://finesse1.xyz.com/finesse/api/ConnectedUsersInfo
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 403: Forbidden 503: Internal Server Error

Example Response:	`<ConnectedUsersInfo>`
	`  <userSummary>`
	`    <desktopUsers>1</desktopUsers>`
	`    <fippaUsers>2</fippaUsers>`
	`    <thirdPartyUsers>0</thirdPartyUsers>`
	`    <totalConnectedUsers>3</totalConnectedUsers>`
	`  </userSummary>`
	`  <uri>/finesse/api/ConnectedUsersInfo</uri>`
	`</ConnectedUsersInfo>`

Example Failure Response:	**Example 1**
	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Internal Server Error</ErrorType>`
	`      <ErrorMessage>Runtime Exception</ErrorMessage>`
	`      <ErrorData></ErrorData>`
	`   </ApiError>`
	`</ApiErrors>`
	Example 2
	`<ApiErrors>`
	`   <ApiError>`
	`      <ErrorType>Service Unavailable</ErrorType>`
	`      <ErrorData>finesse.api.server.outofService</ErrorData>`
	`      <ErrorMessage>SERVER_OUT_OF_SERVICE</ErrorMessage>`
	`   </ApiError>`
	`</ApiErrors>`

ConnectedUsersInfo—Get Connected Users Information

This API retrieves a list of agents who are logged in to the Finesse desktop, with detailed information.

URI:	https://<FQDN>/finesse/api/ConnectedUsersInfo?detailedList=true
Example URI:	https://finesse1.xyz.com/finesse/api/ConnectedUsersInfo?detailedList=true
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Authorization Failure 403: Forbidden 503: Internal Server Error

<table>
<tr><td><b>Example Response:</b></td><td><pre>
<ConnectedUsersInfo>
   <userSummary>
 <desktopUsers>1</desktopUsers>
 <fippaUsers>0</fippaUsers>
 <thirdPartyUsers>0</thirdPartyUsers>
 <totalConnectedUsers>1</totalConnectedUsers>
 <totalConnectedUsersViaProxy>0</totalConnectedUsersViaProxy>
   </userSummary>
   <userDetails>
 <userDetail>
  <loginId>1001002</loginId>
  <userType>DESKTOP</userType>
  <xmppJID>1001002@finesse25.autobot.cvp/desktop</xmppJID>
  <teamName>FunctionalAgents</teamName>
  <extension>1001002</extension>
  <firstName>JOHN</firstName>
  <lastName>SMITH</lastName>
  <connectedViaProxy>false</connectedViaProxy>
  <connectedDuration>2</connectedDuration>
  <connectedHostName>finesse25.autobot.cvp</connectedHostName>
 </userDetail>
   </userDetails>
   <uri>/finesse/api/ConnectedUsersInfo</uri>
</ConnectedUsersInfo>
</pre></td></tr>
<tr><td><b>Example Failure Response:</b></td><td><b>Example 1</b><pre>
<ApiErrors>
   <ApiError>
      <ErrorType>Internal Server Error</ErrorType>
      <ErrorMessage>Runtime Exception</ErrorMessage>
      <ErrorData></ErrorData>
   </ApiError>
</ApiErrors>
</pre><b>Example 2</b><pre>
<ApiErrors>
   <ApiError>
      <ErrorType>Service Unavailable</ErrorType>
      <ErrorData>finesse.api.server.outofService</ErrorData>
      <ErrorMessage>SERVER_OUT_OF_SERVICE</ErrorMessage>
   </ApiError>
</ApiErrors>
</pre></td></tr>
</table>

ConnectedUsersInfo API Parameters

Parameter	Type	Description	Possible Values	Notes
ConnectedUsersInfo	Integer	The container holding information about the agents who are logged-in on to publisher or subscriber.	—	—

Parameter	Type	Description	Possible Values	Notes
userSummary	String	The container which holds summary information of agents who are logged in.	—	—
--> desktopUsers	Integer	Count of Finesse Desktop agents who are logged in.	—	—
-->fippaUsers	Integer	Count of Finesse IP Phone Agents who are logged in.	—	—
-->thirdPartyUsers	Integer	Count of third-party users who are logged in.	—	—
--> totalConnectedUsers	Integer	Total number of connected users in real time.	—	—
userDetails	String	List of user details.	—	—
-->userDetail	String	The container having the list of users, their Jabber Identifiers, and type.	—	—
--> loginId	String	Login Id of the user.	—	—
-->userType	String	Type of the user. • Desktop • Finesse IP Phone Agent • Third-Party	DESKTOP FIPPA THIRDPARTY	—
-->xmppJID	Integer	Full Jabber Identifier (JID) of the user.	—	—
-->teamName	String	Name of the team the agent belong to.	—	—
-->extension	String	Agent's phone extension	—	—
-->first Name	String	First Name of the agent.	—	—
-->last Name	String	Last name of the agent.	—	—

Parameter	Type	Description	Possible Values	Notes
-->connectedDuration	Long	Total duration (in seconds) for which the agent has been logged in.	—	—
-->connectedHostName	String	Finesse host through which the agent is connected.	—	—

ConnectedUsersInfo API Errors

Status	Error Type	Description
401	Authorization Failure	Unauthorized.
403	Forbidden	Only users with administrator privilege allowed to run this.
503	Internal Server Error	Either the Finesse server is out of service, or there is a runtime exception.

Diagnostic Portal

Diagnostic Portal APIs

Diagnostic Portal APIs are primarily to integrate Finesse with the Cisco Prime Contact Center Module and get information about the health of the Finesse system. You can access these APIs only through HTTPS.

Note The Diagnostic Portal APIs are not usable unless Finesse has initially gone IN_SERVICE, after which Finesse can go OUT_OF_SERVICE and the APIs should continue to work.

Diagnostic Portal—Get Performance Information

The Diagnostic Portal—Get Performance Information API allows an administrator to get performance information to a Diagnostic Portal object.

URI:	https://FQDN/finesse-dp/rest/DiagnosticPortal/GetPerformanceInformation
Example URI:	https://finesse1.xyz.com/finesse-dp/rest/DiagnosticPortal/GetPerformanceInformation
Security Constraints:	A user must be signed in as an administrator to use this API.
HTTP Method:	GET

Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success **Note** All requests that reach the Finesse Diagnostic Portal web application return a 200 response. However, requests that are not successfully handled return XML that includes an error code and optionally, an error string. 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Successful Response:

```
<dp:GetPerformanceInformationReply
xmlns:dp="http://www.cisco.com/vtg/diagnosticportal" ReturnCode="0">
<dp:Schema Version="1.0"/>
   <dp:PerformanceInformation>
      <dp:PropertyList>
         <dp:Property Name="Tomcat/Heap Memory Utilized"
Value="1739233744"/>
         <dp:Property Name="Tomcat/Non Heap Memory Utilized"
Value="269373496"/>
         <dp:Property Name="Active Totals/Logged In Agents" Value="1"/>
         <dp:Property Name="Active Totals/Current Calls" Value="0"/>
         <dp:Property Name="Running Totals/Calls Received or Initiated"
Value="17803"/>
         <dp:Property Name="Running Totals/Calls Failed" Value="0"/>
         <dp:Property Name="CTI Statistics/Events In Queue" Value="0"/>
         <dp:Property Name="CTI Statistics/Outgoing Responses Queue"
Value="0"/>
         <dp:Property Name="CTI Statistics/Decoding Responses Queue"
Value="0"/>
         <dp:Property Name="Tomcat/Average Request Process Time"
Value="0"/>
         <dp:Property Name="Tomcat/Longest Request Process Time"
Value="174"/>
         <dp:Property Name="Tomcat/Thread Count" Value="323"/>
         <dp:Property Name="Tomcat/Peak Thread Count" Value="481"/>
         <dp:Property Name="Average System Load" Value="0.12"/>
      </dp:PropertyList>
   </dp:PerformanceInformation>
</dp:GetPerformanceInformationReply>
```

Note From the Cisco Finesse Release 12.5(1), CTI Statistics or Incoming Responses Queue is removed due to the architecture changes in the CTI event processing.

Example Failure Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<dp:GetProductLicenseReply ReturnCode="1" ErrorString="License file
license.txt could not be
  read" xmlns:dp="http://www.cisco.com/vtg/diagnosticportal">
<dp:Schema Version="1.0"/>
</dp:GetProductLicenseReply>
```

Diagnostic Portal—Get Product Version

This API allows an administrator to get product version information for Finesse.

URI:	https://FQDN/finesse-dp/rest/DiagnosticPortal/GetProductVersion
Example URI:	https://finesse1.xyz.com/finesse-dp/rest/DiagnosticPortal/GetProductVersion
Security Constraints:	A user must be signed in as an administrator to use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success **Note** All requests that reach the Finesse Diagnostic Portal web application return a 200 response. However, requests that are not successfully handled return XML that includes an error code and optionally, an error string. 401: Authorization Failure 403: Forbidden 404: Not Found 500: Internal Server Error

Successful Response:	`<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`
	`<dp:GetProductVersionReply ReturnCode="0">`
	`   xmlns:dp="http://cisco.com/vtg/diagnosticportal" ReturnCode="0"`
	`   <dp:Schema Version="1.0"/>`
	`   <dp:ProductVersion Name="Cisco Finesse" Major="12" Minor="6"`
	`Maintenance="1" VersionString="12.6(1)"/>`
	`   <dp:ComponentVersionList/>`
	`</dp:GetProductVersionReply>`

Example Failure Response:	`<?xml version="1.0" encoding="UTF-8" ?>`
	`<dp:GetProductLicenseReply ReturnCode="1" ErrorString="License file`
	` license.txt could not be read" xmlns:dp="http://www.cisco.com/vtg/`
	` diagnosticportal">`
	`<dp:Schema Version="1.0"/>`
	`</dp:GetProductLicenseReply>`

Diagnostic Portal API Errors

Status	Error Type	Description
401	Authorization Error	The user is not authorized to access this API.

Status	Error Type	Description
403	Forbidden	The user is not authorized to use the API (the user is not an administrator).
404	Not Found	The resource is not found (for example, the DiagnosticPortal has been deleted).
500	Internal Server Error	Any runtime exception is caught and responded with this error.

RuntimeConfigInfo

RuntimeConfigInfo APIs

RuntimeConfigInfo—Get

This API allows an administrator to access run time information.

URI:	https://<FQDN>/finesse/api/RuntimeConfigInfo
Example URI:	https://finesse1.xyz.com/finesse/api/RuntimeConfigInfo
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 403: Forbidden 500: Internal Server Error

Example Response:	
	```
	<RuntimeConfigInfo>
	 <activeDialogCount>0</activeDialogCount>
	 <activeTaskCount>0</activeTaskCount>
	 <averageConfiguredMediaPerAgent>0</averageConfiguredMediaPerAgent>
	 <averageLoggedInMediaPerAgent>0</averageLoggedInMediaPerAgent>
	 <averageSkillGroupCountPerAgent>0</averageSkillGroupCountPerAgent>
	 <connectedUsersInfo>
	  <userSummary>
	   <desktopUsers>3</desktopUsers>
	   <fippaUsers>1</fippaUsers>
	   <thirdPartyUsers>0</thirdPartyUsers>
	  </userSummary>
	 </connectedUsersInfo>
	 <maxSkillGroupCountPerAgent>0</maxSkillGroupCountPerAgent>
	 <timeToInService>11</timeToInService>
	 <totalLoggedInAgentsInNode>0</totalLoggedInAgentsInNode>
	 <uniqueConfiguredSkillGroups>0</uniqueConfiguredSkillGroups>
	 <uri>/finesse/api/RuntimeConfigInfo</uri>
	</RuntimeConfigInfo>
	```

Example Failure Response:	
	```
	<ApiErrors>
	   <ApiError>
	      <ErrorType>Authorization Failure</ErrorType>
	      <ErrorMessage>UNAUTHORIZED</ErrorMessage>
	      <ErrorData>jsmith</ErrorData>
	   </ApiError>
	</ApiErrors>
	```

RuntimeConfigInfo API Parameters

Parameter	Type	Description	Possible Values	Notes
activeDialogCount	Integer	The count of active calls present in the node.	—	—
activeTaskCount	Integer	The count of active tasks present in the node.	—	—
averageConfiguredMediaPerAgent	Integer	The average of the configured media channels for the logged in agents (voice). For example, Agent 1 has logged in to the voice channel and has configured for voice and chat. Agent 2 has logged in to the voice channel and has configured for voice, email, and chat. Result is (2+3)/ 2 = 2	—	This parameter is not applicable for Unified CCX. However, the value is considered as 1.

Parameter	Type	Description	Possible Values	Notes
averageLoggedInMediaPerAgent	Integer	The average of the logged in media channels by the agent who has logged in (voice). For example, Agent 1 has logged in to the voice channel and chat. Agent 2 has logged in the voice channel along with email. Result is (2+2)/ 2 = 2	—	This parameter is not applicable for Unified CCX. However, the value is considered as 1.
averageSkillGroupCountPerAgent	Integer	The count of the average configured skill groups among all the logged in agents for that node. For example, • Agent 1—3 configured skill groups • Agent 2—2 configured skill groups • Agent 3—1 configured skill groups Result is (3+2+1)/ 3 = 2	—	—
connectedUsersInfo	Collection	Information of the agents logged in to Cisco Finesse.	—	—
→ userSummary	Collection	Summary information of the agents logged in to Cisco Finesse.	—	—
→ desktopUsers	Integer	The number of desktop agents logged in to Cisco Finesse.	—	—
→fippaUsers	Integer	The number of Finesse IPPA agents logged in to Cisco Finesse.	—	—
→ thirdPartyUsers	Integer	The number of third-party agents logged in to Cisco Finesse.	—	—

Parameter	Type	Description	Possible Values	Notes
maxSkillGroupCountPerAgent	Integer	The count of the maximum configured skill groups among all the logged in agents for that node. For example, • Agent 1—3 configured skill groups • Agent 2—2 configured skill groups • Agent 3—1 configured skill groups Result is 3 (maximum count)	—	—
timeToInService	Integer	The time taken by Cisco Finesse to connect with the CTI server in seconds.	—	—
totalLoggedInAgentsInNode	Integer	The count of the logged in agents for the voice channel only.	—	—
uniqueConfiguredSkillGroups	Integer	The count of the unique skill groups among all the logged in agents for that node.	—	—
uri	String	The URI to get a new copy of the RuntimeConfigInfo object.	—	—

RuntimeConfigInfo API Errors

Status	Error Type	Description
401	Authorization Failure	Unauthorized (for example, the user is not yet authenticated in the Web Session).
403	Forbidden	The user attempted to run the API against the secondary Cisco Finesse server. Configuration APIs cannot be run against the secondary Cisco Finesse server.
500	Internal Server Error	Any runtime exception is caught and responded with this error.

Locked Out Users

This API lists the locked-out users.

URI:	https://<FQDN>/finesse/api/LockedOutUsers
Example URI:	https://finesse1.xyz.com/finesse/api/LockedOutUsers
Security Constraints:	Only administrators can use this API.
HTTP Method:	GET
Content Type:	—
Input/Output Format:	XML
HTTP Request:	—
HTTP Response:	200: Success 401: Unauthorized 403: Forbidden 500: Internal Server Error

Example Response:	`<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`
	`<LockedOutUsers>`
	`   <LockedOutUser>`
	`      <loginId>1001004</loginId>`
	`      <firstName>AGENT</firstName>`
	`      <lastName>1001004</lastName>`
	
	`<lastFailureTime>1634810571512</lastFailureTime>`
	
	`      <failureCount>5</failureCount>`
	`      <lockStatus>true</lockStatus>`
	`   </LockedOutUser>`
	`   <LockedOutUser>`
	`      <loginId>1001003</loginId>`
	`      <firstName>AGENT</firstName>`
	`      <lastName>1001003</lastName>`
	
	`<lastFailureTime>1634810557306</lastFailureTime>`
	
	`      <failureCount>5</failureCount>`
	`      <lockStatus>true</lockStatus>`
	`   </LockedOutUser>`
	`   <uri>/finesse/api/LockedOutUsers</uri>`
	`</LockedOutUsers>`

Example Failure Response:	
	```
	<ApiErrors>
	   <ApiError>
	      <ErrorType>Service
	Unavailable</ErrorType>

	<ErrorData>finesse.api.server.outofService</ErrorData>


	<ErrorMessage>SERVER_OUT_OF_SERVICE</ErrorMessage>

	   </ApiError>
	</ApiErrors>
	```

CHAPTER 6

Cisco Finesse Notifications

About Cisco Finesse Notifications

The Cisco Finesse Web Service sends notifications to clients that subscribe to that class of resource.

For example, a client that is subscribed to *User* notifications receives a notification when an agent signs in or out of the Finesse desktop, information about an agent changes, or an agent's state changes.

Note The preceding example illustrates some cases where notifications are sent. It is not intended to be an exhaustive list.

Note Notification payloads are XML-encoded. If these payloads contain any special XML characters, you must ensure that the client decodes this information correctly before processing it further.

Notification Frequency

Finesse publishes notifications when a change occurs in the resource characteristics.

Subscription Management

Finesse clients can interface directly with the Cisco Finesse Notification Service to send subscribe and unsubscribe requests. Clients subscribe to notification feeds published to their respective nodes (such as /finesse/api/User/1000) by following the XEP-0060 standard.

Each agent is automatically subscribed to the following notification feeds, where {id} represents the agent ID for that agent:

- User - /finesse/api/User/{id}
- Dialogs - /finesse/api/User/{id}/Dialogs
- Media - /finesse/api/User/{id}/Media/{mrd-id}

- SystemInfo - /finesse/api/SystemInfo

To receive notifications for feeds to which they are not automatically subscribed, clients must explicitly subscribe to the node on which the notifications are published. For example, agent state change notifications for all agents on a specific team are published to the node /finesse/api/Team/{id}/Users. Clients must request a subscription to this node to receive notifications on this feed.

To avoid increasing notification traffic for other users, use a full JID (username@domain/resource) when making explicit subscriptions.

Make sure to unsubscribe to any explicit subscriptions before disconnecting the XMPP session. Any subscriptions that are left behind persist on that node in the Cisco Finesse Notification Service.

The following example shows how to subscribe to agent state change notifications for a specific team:

```
<iq type='set'
    from='CharlesNorrad@finesse-server.cisco.com'
    to='pubsub.finesse-server.cisco.com'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe
        node='/finesse/api/Team/TheA/Users'
        jid='ChuckieNorrad@finesse-server.cisco.com'/>
  </pubsub>
</iq>
```

The following example shows how to unsubscribe to agent state change notifications for a specific team:

```
<iq type='set'
    from='ChuckieNorrad@finesse-server.cisco.com'
    to='pubsub.finesse-server.cisco.com'
    id='unsub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
     <unsubscribe
         node='/finesse/api/Team/TheA/Users'
         jid='userid@finesse-server.cisco.com'/>
  </pubsub>
</iq>
```

Perform a GET using the SystemInfo API (https://<server>/finesse/api/SystemInfo) to obtain connection details. The returned payload provides the domain and pubsub addresses used to interact with the Cisco Finesse Notification Service.

```
<SystemInfo>
   <status>IN_SERVICE</status>
   <xmppDomain>xmppserver.cisco.com</xmppDomain>
   <xmppPubSubDomain>pubsub.xmppserver.cisco.com</xmppPubSubDomain>
</SystemInfo>
```

Users are identified in the following manner: userid@xmppserver.cisco.com

Stanzas are sent to the pubsub domain (pubsub.xmppserver.cisco.com).

Clients should ensure that any subscriptions that are no longer required are cleaned up.

Subscription Persistence

All subscriptions are stored in a database and persist through the following shutdown events:

- Finesse experiences a CTI failover.
- The Cisco Finesse Notification Service restarts.
- Cisco Finesse Tomcat restarts.

In each of the preceding events, the client does not need to resubscribe to explicit subscriptions.

However, subscriptions do not persist across multiple Finesse servers. If a client fails over to an alternate Finesse server, that client must resubscribe to any explicit subscriptions.

Resources

User Notification

Finesse sends a User notification when information about a user changes.

Format:	XML
Node:	/finesse/api/User/{id}
Source:	/finesse/api/User/{id}
Data:	User

Payload:	`<Update>`
	`   <event>{put|delete}</event>`
	`   <source>/finesse/api/User/{id}</source>`
	`   <data>`
	`      <user>`
	`      <!-- full User object -->`
	`      </user>`
	`   </data>`
	`</Update>`

Sample Notification Payload:	`<Update>`
	`   <event>put</event>`
	`   <source>/finesse/api/User/csmith</source>`
	`   <data>`
	`      <User>`
	`        <dialogs>/finesse/api/User/1001001/Dialogs</dialogs>`
	`        <extension></extension>`
	`        <firstName>AGENT</firstName>`
	`        <lastName>1001001</lastName>`
	`        <loginId>1001001</loginId>`
	`        <loginName>agent1</loginName>`
	`        <pendingState></pendingState>`
	`        <reasonCodeId>2</reasonCodeId>`
	`        <ReasonCode>`
	`          <uri>/finesse/api/ReasonCode/{id}</uri>`
	`          <code>10</code>`
	`          <label>Team Meeting</label>`
	`        </ReasonCode>`
	`        <settings>`
	`          <wrapUpOnIncoming>OPTIONAL</wrapUpOnIncoming>`
	`          <wrapUpOnOutgoing>REQUIRED</wrapUpOnOutgoing>`
	`        </settings>`
	`        <roles>`
	`          <role>Agent</role>`
	`        </roles>`
	`        <state>LOGOUT</state>`
	`        <stateChangeTime></stateChangeTime>`
	`        <teamId>5000</teamId>`
	`        <teamName>FunctionalAgents</teamName>`
	`        <uri>/finesse/api/User/1001001</uri>`
	`      </User>`
	`   </data>`
	`</Update>`

Notification Triggers:	• Addition of a user Addition of a user • Deletion of a user • State change • First or last name change • Role change • Pending state change

Dialog Notification

Finesse sends a Dialog notification when information (or an action) changes for a call to which the user belongs or when the user adds or removes a dialog.

For the purpose of notifications, the fromAddress and toAddress parameters of the Dialog object are defined as follows:

- fromAddress: The extension of the caller who initiated the original call. If an unmonitored caller placed the call, the fromAddress is the unmonitored caller's extension. If an agent placed the call, the fromAddress is the agent's extension. For an Outbound Option Dialer call, the fromAddress is the extension of the agent on the outbound call. For a reservation call in Preview Outbound mode, the fromAddress is the dialer port. .

For a reservation call in Direct Preview Outbound mode, the fromAddress is the dialer port.

- toAddress: The dialed number of the original call. If the caller calls a route point, the toAddress is the route point. If the caller calls an agent directly, the toAddress is the agent's extension. For an Outbound Option Dialer call, the toAddress is the customer phone number called by the dialer. For a reservation call in Outbound Option Preview mode, the toAddress is the extension of the agent who received the call.

  For a reservation call in Direct Preview Outbound mode, the toAddress is the extension of the agent on the outbound call.

When a call is transferred, the fromAddress and toAddress in subsequent dialog notifications are those of the surviving call. For example, if an agent who is on a call places a consult call and then transfers the original call, the fromAddress and toAddress in the subsequent dialog notifications are those of the original call because the original call is the surviving call. However, if the agent puts the consult call on hold, retrieves the original call, and then transfers the consult call, the fromAddress and toAddress in subsequent dialog notifications are those of the consult call. In this case, the consult call is the surviving call.

When an agent who is on a call places a consult call, the original call will be on hold and the consult call will be active. Once the call is complete where the agent either transfers or places the call on conference, the surviving call's dialog notifications will contain the dropped call's dialog id in the secondary id field.

During Dialog notifications, there are two types of notifications that get sent to the Dialog node.

- When a dialog is added or removed from the Dialog collection of the user.

Format:	XML
Node:	/finesse/api/User/{id}/Dialogs
Source:	/finesse/api/User/{id}/Dialogs (when a Dialog is added or removed from the Dialog collection for the user)
Data:	Dialogs

Payload:	`<Update>`
	`  <data>`
	`    <dialogs>`
	`      <Dialog>`
	`        <!-- full Dialog object -->`
	`      </Dialog>`
	`    </dialogs>`
	`  </data>`
	`  <event>{POST|DELETE}</event>`
	`  <requestId>xxxxxxxxx</requestId>`
	`  <source>/finesse/api/User/{id}/Dialogs</source>`
	`</Update>`

Sample Notification Payload:	

	```
	<Update>
	    <data>
	        <dialogs>
	            <Dialog>
	                <associatedDialogUri></associatedDialogUri>
	                <fromAddress>1112554</fromAddress>
	                <id>2130715746</id>
	                <secondaryId>2130715747</secondaryId>
	                <mediaProperties>
	                    <mediaId>1</mediaId>
	                    <DNIS>90101</DNIS>
	                    <callType>CONSULT</callType>
	                    <dialedNumber>90101</dialedNumber>
	                    <outboundClassification></outboundClassification>
	                    <callvariables>
	                        <CallVariable>
	                            <name>callVariable1</name>
	                            <value>1</value>
	                        </CallVariable>
	                         ....
	                        <CallVariable>
	                            <name>callVariable2</name>

	<value>0123456789ABCDEFGHIJ0123456789ABCDEFGHIJ</value>
	                        </CallVariable>
	                        <CallVariable>
	                            <name>callVariable3</name>

	<value>0123456789ABCDEFGHIJ0123456789ABCDEFGHIJ</value>
	                        </CallVariable>
	                        <CallVariable>
	                            <name>callVariable4</name>

	<value>0123456789ABCDEFGHIJ0123456789ABCDEFGHIJ</value>
	                        </CallVariable>
	                        <CallVariable>
	                            <name>callVariable5</name>

	<value>0123456789ABCDEFGHIJ0123456789ABCDEFGHIJ</value>
	                        </CallVariable>
	                        <CallVariable>
	                            <name>callVariable6</name>

	<value>0123456789ABCDEFGHIJ0123456789ABCDEFGHIJ</value>
	                        </CallVariable>
	                        <CallVariable>
	                            <name>callVariable7</name>

	<value>0123456789ABCDEFGHIJ0123456789ABCDEFGHIJ</value>
	                        </CallVariable>
	                        <CallVariable>
	                            <name>callVariable8</name>

	<value>0123456789ABCDEFGHIJ0123456789ABCDEFGHIJ</value>
	                        </CallVariable>
	                        <CallVariable>
	                            <name>callVariable9</name>

	<value>0123456789ABCDEFGHIJ0123456789ABCDEFGHIJ</value>
	                        </CallVariable>
	                        <CallVariable>
	                            <name>callVariable10</name>

	<value>0123456789ABCDEFGHIJ0123456789ABCDEFGHIJ</value>
	```

	```
	                        </CallVariable>
	                    </callvariables>
	                    <queueNumber>5022</queueNumber>
	                    <queueName>UCM_PIM.Func.Agents.SG</queueName>
	                    <callKeyCallId>217</callKeyCallId>
	                    <callKeySequenceNum>1</callKeySequenceNum>
	                    <callKeyPrefix>152018</callKeyPrefix>
	                </mediaProperties>
	                <mediaType>Voice</mediaType>
	                <participants>
	                    <Participant>
	                        <actions>
	                            <action>UPDATE_CALL_DATA</action>
	                            <action>DROP</action>
	                        </actions>
	                        <mediaAddress>1112554</mediaAddress>
	                        <mediaAddressType>AGENT_DEVICE</mediaAddressType>

	                        <startTime>2016-05-03T21:49:36.512Z</startTime>
	                        <state>INITIATING</state>
	                        <stateCause></stateCause>

	<stateChangeTime>2016-05-03T21:49:36.512Z</stateChangeTime>
	                    </Participant>
	                </participants>
	                <state>INITIATING</state>
	                <toAddress>90101</toAddress>
	                <uri>/finesse/api/Dialog/2130715746</uri>
	            </Dialog>
	        </dialogs>
	    </data>
	    <event>POST</event>
	    <requestId>edc7064f-1178-11e6-8bd0-005056000005</requestId>
	    <source>/finesse/api/User/112554/Dialogs</source>
	</Update>
	```
Notification Triggers:	• Incoming call • Ending a call

- When dialog properties associated with the specified Dialog id is modified.

Format:	XML
Node:	/finesse/api/User/{id}/Dialogs
Source:	/finesse/api/Dialog/{id} (when a Dialog within the Dialogs collection for the user is modified)
Data:	Dialog

Payload:	
	```
	<Update>
	  <data>
	    <dialog>
	      <!-- full Dialog object -->
	    </dialog>
	  </data>
	  <event>{PUT}</event>
	  <requestId>xxxxxxxxx</requestId>
	  <source>/finesse/api/Dialog/16804377</source>
	</Update>
	```

Sample Notification Payload:	

```
<Update>
    <data>
        <dialog>
            <associatedDialogUri></associatedDialogUri>
            <fromAddress>1081001</fromAddress>
            <id>16804377</id>
            <mediaProperties>
                <mediaId>1</mediaId>
                <DNIS>1081002</DNIS>
                <callType>AGENT_INSIDE</callType>
                <callvariables>
                    <CallVariable>
                        <name>callVariable1</name>
                        <value></value
                <queueNumber>5022</queueNumber>
                <queueName>UCM_PIM.Func.Agents.SG</queueName>
                <callKeyCallId>217</callKeyCallId>
                <callKeySequenceNum>1</callKeySequenceNum>
                <callKeyPrefix>152018</callKeyPrefix>
                <dialedNumber>1081002</dialedNumber>
                </mediaProperties>
                <mediaType>Voice</mediaType>
                <participants>
                    <Participant>
                        <actions>
                            <action>TRANSFER_SST</action>
                            <action>CONSULT_CALL</action>
                            <action>HOLD</action>
                            <action>UPDATE_CALL_DATA</action>
                            <action>SEND_DTMF</action>
                            <action>DROP</action>
                        </actions>
                        <mediaAddress>1081001</mediaAddress>

<mediaAddressType>AGENT_DEVICE</mediaAddressType>
                     <startTime>2014-02-04T15:33:16.653Z</startTime>

                        <state>ACTIVE</state>
                        <stateCause></stateCause>

<stateChangeTime>2014-02-04T15:33:16.653Z</stateChangeTime>
                    </Participant>
                    <Participant>
                        <actions>
                            <action>UPDATE_CALL_DATA</action>
                            <action>DROP</action>
                            <action>RETRIEVE</action>
                        </actions>
                        <mediaAddress>1081002</mediaAddress>

<mediaAddressType>AGENT_DEVICE</mediaAddressType>
                     <startTime>2014-02-04T15:33:16.653Z</startTime>

                        <state>HELD</state>
                        <stateCause></stateCause>

<stateChangeTime>2014-02-04T15:33:27.584Z</stateChangeTime>
                    </Participant>
                </participants>
                <state>ACTIVE</state>
                <toAddress>1081002</toAddress>
                <uri>/finesse/api/Dialog/16804377</uri>
        </dialog>
    </data>
```

	`<event>PUT</event>`
	`<requestId>xxxxxxxxx</requestId>`
	`<source>/finesse/api/Dialog/16804377</source>`
	`</Update>`

Notification Triggers:	• Modification of participant state (for example, when a participant answers or hangs up a call) • A new participant on the call • Modification of the call data or actions

Dialogs/Media Notification

Finesse sends a Dialogs/Media notification when information (or an action) changes for a nonvoice dialog to which the user belongs.

Important For an interruptible Media Routing Domain configured to accept interrupts, Finesse sends only a Media state change when an agent is interrupted in that MRD. It does not send Dialogs/Media notifications with the action list modified to reflect the fact that actions not permitted on the tasks in that media. The state change is the only indication to the Finesse applications that no actions are allowed on the interrupted dialogs.

During Dialog notifications, there are two types of notifications that get sent to the Dialog node.

- When a dialog is added or removed from the Dialog collection of the user.

Format:	XML
Node:	/finesse/api/User/{id}/Dialogs/Media
Source:	/finesse/api/User/{id}/ Media/{mrdId}/Dialogs (when a Dialog is added or removed from the Dialog collection for the user, for example offered or closed)
Data:	Dialogs

Payload:	`<Update>`
	`  <data>`
	`    <dialogs>`
	`      <Dialog>`
	`        <!-- full Dialog object -->`
	`      </Dialog>`
	`    </dialogs>`
	`  </data>`
	`  <event>{POST|DELETE}</event>`
	`  <requestId>xxxxxxxxx</requestId>`
	`  <source>/finesse/api/User/{id}/Media{mrdld}/Dialogs</source>`
	`</Update>`

Sample Notification Payload

```
<Update>
    <data>
        <dialogs>
            <Dialog>

<associatedDialogUri>/finesse/api/Dialog/3216_5432_1</associatedDialogUri>

                <id>1234_5423_1</id>
                <mediaType>Cisco_Chat_MRD</mediaType>
                <mediaProperties>
                    <mediaId>5002</mediaId>
                    <dialedNumber></dialedNumber>
                    <callvariables>
                        <CallVariable>
                            <name>callVariable1</name>
                            <value>Chuck Smith</value>
                        </CallVariable>
                        <CallVariable>
                            <name>callVariable2</name>
                            <value>Cisco Systems, Inc.</value>
                        ...Other CallVariables ...
                        </callvariables>
                        <queueNumber>5022</queueNumber>

<queueName>UCM_PIM.Func.Agents.SG</queueName>
                        <callKeyCallId>217</callKeyCallId>
                       <callKeySequenceNum>1</callKeySequenceNum>

                        <callKeyPrefix>152018</callKeyPrefix>
                    </mediaProperties>
                    <participants>
                        <Participant>
                            <actions>
                                <action>ACCEPT</action>
                            </actions>
                            <mediaAddress>1001001</mediaAddress>

<startTime>2015-11-19T06:04:27.864Z</startTime>
                            <state>OFFERED</state>

<stateChangeTime>2015-11-19T06:04:27.864Z</stateChangeTime>
                        </Participant>
                    </participants>
                    <state>OFFERED</state>
                    <uri>/finesse/api/Dialog/1234_5423_1</uri>
            </Dialog>
        </dialogs>
    </data>
    <event>POST</event>
    <requestId>xxxxxxxxx</requestId>
   <source>/finesse/api/User/10010012/Media{5002}/Dialogs</source>
</Update>
```

Notification Triggers:	• Incoming dialog

• When dialog properties associated with the specified Dialog id is modified.

Format:	XML
Node:	/finesse/api/User/{id}/Dialogs/Media

Source:	/finesse/api/Dialog/{id} (when a Dialog within the Dialogs collection for the user is modified, for example accepted, started, paused, or wrapped up)
Data:	Dialog

Payload:	`<Update>`
	`  <data>`
	`    <dialog>`
	`       <!-- full Dialog object -->`
	`    </dialog>`
	`  </data>`
	`  <event>{PUT}</event>`
	`  <requestId>xxxxxxxxx</requestId>`
	`  <source>/finesse/api/Dialogs{id}</source>`
	`</Update>`

Sample Notification Payload	
	```
	Update>
	    <data>
	        <dialog>
	            <associatedDialogUri/>
	            <id>151705_33542697_1</id>
	            <mediaProperties>
	                <mediaId>5000</mediaId>
	                <dialedNumber>mark_test_dn</dialedNumber>
	                <callvariables>
	                    <CallVariable>
	                        <name>callVariable1</name>
	                        <value>cv1_value</value>
	                    </CallVariable>
	                    <CallVariable>
	                        <name>callVariable2</name>
	                        <value>cv2_value</value>
	                    </CallVariable>
	                    <CallVariable>
	                        <name>user.finesse.ecc1</name>
	                        <value>ecc1</value>
	                    </CallVariable>
	                </callvariables>
	                <queueNumber>5022</queueNumber>
	                <queueName>UCM_PIM.Func.Agents.SG</queueName>
	                <callKeyCallId>217</callKeyCallId>
	                <callKeySequenceNum>1</callKeySequenceNum>
	                <callKeyPrefix>152018</callKeyPrefix>
	            </mediaProperties>
	            <mediaType>Cisco_Chat_MRD</mediaType>
	            <participants>
	                <Participant>
	                    <actions>
	                        <action>START</action>
	                        <action>CLOSE</action>
	                        <action>TRANSFER</action>
	                    </actions>
	                    <mediaAddress>1001010</mediaAddress>
	                   <startTime>2016-05-10T20:25:12.302Z</startTime>

	                    <state>ACCEPTED</state>

	<stateChangeTime>2016-05-10T20:25:17.372Z</stateChangeTime>
	                </Participant>
	            </participants>
	            <state>ACCEPTED</state>
	            <uri>/finesse/api/Dialog/151705_33542697_1</uri>
	        </dialog>
	    </data>
	    <event>PUT</event>
	    <requestId/>
	    <source>/finesse/api/Dialog/{id}</source>
	</Update>
	```

Notification Triggers:	• Modification of participant state (for example, when a participant accepts or closes a dialog)

Dialog CTI Error Notification

Call operations performed on a dialog (such as MAKE_CALL, HOLD, RETRIEVE, ANSWER, END, TRANSFER, CONSULT, and CONFERENCE) may result in CTI errors. The notification system sends these errors as asynchronous updates. Error notifications include the error type and the CTI error code and error constant. The error type is “Call Operation Failure”.

Format:	XML
Node:	/finesse/api/User/{id}/Dialogs
Source:	/finesse/api/Dialog/{id}
Data:	apiErrors

Payload:	`<Update>`
	`    <data>`
	`        <apiErrors>`
	`            <apiError>`
	`                <errorData>[CTI Error Code]</errorData>`
	`               <errorMessage>[CTI Error Constant]</errorMessage>`
	
	`                <errorType>Call Operation Failure</errorType>`
	`            </apiError>`
	`        </apiErrors>`
	`    </data>`
	`    <event>PUT</event>`
	`    <requestId></requestId>`
	`    <source>/finesse/api/Dialog/[ID]</source>`
	`</Update>`

Sample Notification Payload	`<Update>`
	`   <data>`
	`      <apiErrors>`
	`          <apiError>`
	`              <errorData>34</errorData>`
	
	`<errorMessage>CF_RESOURCE_OUT_OF_SERVICE</errorMessage>`
	`              <errorType>Call Operation Failure</errorType>`
	`          </apiError>`
	`      </apiErrors>`
	`   </data>`
	`   <event>PUT</event>`
	`   <requestId></requestId>`
	`   <source>/finesse/api/Dialog/12345</source>`
	`</Update>`

Notification Triggers:	The notification system delivers this error notification if call operations on a Dialog (such as MAKE_CALL, HOLD, RETRIEVE, ANSWER, END, TRANSFER, CONSULT, and CONFERENCE) result in a CTI error

Asynchronous Errors

Note When accessing the Finesse REST API through the Finesse JavaScript library, asynchronous errors have a status code of 400. When receiving the asynchronous error directly through XMPP, the error message has the format described in the description above for Dialog CTI Error Notification.

ErrorType	Reason	Deployment Type
Call Operation Failure	Attempt to exceed maximum allowed conference participants.	Unified CCE

Team Notification

Finesse sends a team notification when the agent name or agent state changes for an agent who belongs to that team.

Format:	XML
Node:	/finesse/api/Team/{id}/Users
Source:	/finesse/api/User/{id}
Data:	Summary version of the User object
Payload:	See below

```
<Update>
    <event>{put}</event>
    <source>/finesse/api/User/{id}</source>
    <requestId>xxxxxxxxx</requestId>
    <data>
        <user>
            <uri>/finesse/api/User/{id}</uri>
            <loginId>{id}</loginId>
            <firstName>Jack</firstName>
            <lastName>Brown</lastName>
            <state>NOT_READY</state>

<stateChangeTime>2012-03-01T17:58:21.123Z</stateChangeTime>
            <ReasonCode>
                <uri>finesse/api/ReasonCode/1</uri>
                <code>10</code>
                <label>Team Meeting</label>
                <category>NOT_READY</category>
                <id>1</id>
            </ReasonCode>
        </user>
    </data>
</Update>
```

Sample Notification Payload:	`<Update>`
	`    <event>put</event>`
	`    <source>/finesse/api/Team/1004</source>`
	`    <requestId>xxxxxxxxx</requestId>`
	`    <data>`
	`        <team>`
	`            <uri>/finesse/api/Team/1004</uri>`
	`            <id>1004</id>`
	`            <name>Shiny</name>`
	`            <users>`
	`                <User>`
	`                    <uri>/finesse/api/User/1234</uri>`
	`                    <loginId>1004</loginId>`
	`                    <firstName>Charles</firstName>`
	`                    <lastName>Norrad</lastName>`
	`                    <pendingState></pendingState>`
	`                    <state>LOGOUT</state>`
	
	`<stateChangeTime>2012-03-01T17:58:21.123Z</stateChangeTime>`
	`                </User>`
	`                <User>`
	`                    <uri>/finesse/api/User/9876</uri>`
	`                    <loginId>9876</loginId>`
	`                    <firstName>Jack</firstName>`
	`                    <lastName>Brown</lastName>`
	`                    <state>NOT_READY</state>`
	
	`<stateChangeTime>2012-03-01T17:58:21.134Z</stateChangeTime>`
	`                    <ReasonCode>`
	`                        <uri>/finesse/api/ReasonCode/1</uri>`
	`                        <code>10</code>`
	`                        <label>Team Meeting</label>`
	`                        <category>NOT_READY</category>`
	`                        <id>1</id>`
	`                    </ReasonCode>`
	`                </User>`
	`           ... other users ...`
	`            </users>`
	`        </team>`
	`    </data>`
	`</Update>`

Notification Triggers:	• Agent name is changed for an agent who belongs to the team • Agent state is changed for an agent who belongs to the team

Queue Notifications

Finesse sends a queue notification every 10 seconds (if queue statistics change).

Note Finesse sends notifications for this node only for a stand-alone Finesse deployment with Unified CCE. Notifications for this node are not sent for a coresident Finesse deployment with Unified CCX.

Format:	XML
Node:	/finesse/api/Queue/{id}
Source:	/finesse/api/Queue/{id}

Data:	Queue object

Payload (PUT):

```
<Update>
    <event>{put}</event>
    <source>/finesse/api/Queue/{id}</source>
    <requestId>xxxxxxxxx</requestId>
    <data>
        <Queue>
            <uri>/finesse/api/Queue/{id}</uri>
            <name>Sales</name>
            <statistics>
                <callsInQueue>3</callsInQueue>

<startTimeOfLongestCallInQueue>2012-02-15T17:58:21Z</startTimeOfLongestCallInQueue>

                <agentsReady>1</agentsReady>
                <agentsNotReady>2</agentsNotReady>
                <agentsTalkingInbound>3</agentsTalkingInbound>
                <agentsTalkingOutbound>4</agentsTalkingOutbound>
                <agentsTalkingInternal>5</agentsTalkingInternal>
                <agentsWrapUpNotReady>6</agentsWrapUpNotReady>
                <agentsWrapUpReady>7</agentsWrapUpReady>
            </statistics>
        </Queue>
    </data>
</Update>
```

Payload (DELETE):

```
<Update>
    <event>{delete}</event>
    <source>/finesse/api/Queue/{id}</source>
    <requestId></requestId>
    <data>
       <Queue>
           <uri>/finesse/api/Queue/{id}</uri>
        </Queue>
     </data>
</Update>
```

Sample Notification Payload (PUT):

```
<Update>
    <event>put</event>
    <source>/finesse/api/Queue/1004</source>
    <requestId>xxxxxxxxx</requestId>
    <data>
        <Queue>
            <uri>/finesse/api/Queue/1004</uri>
            <name>Sales</name>
            <statistics>
                <callsInQueue>3</callsInQueue>

<startTimeOfLongestCallInQueue>2012-02-15T17:58:21Z</startTimeOfLongestCallInQueue>

                <agentsReady>1</agentsReady>
                <agentsNotReady>2</agentsNotReady>
                <agentsTalkingInbound>3</agentsTalkingInbound>
                <agentsTalkingOutbound>4</agentsTalkingOutbound>
                <agentsTalkingInternal>5</agentsTalkingInternal>
                <agentsWrapUpNotReady>6</agentsWrapUpNotReady>
                <agentsWrapUpReady>7</agentsWrapUpReady>
            </statistics>
        </Queue>
    </data>
</Update>
```

Sample Notification Payload (DELETE):	`<Update>`
	`     <event>delete</event>`
	`     <source>/finesse/api/Queue/1004</source>`
	`     <requestId></requestId>`
	`     <data>`
	`         <Queue>`
	`            <uri>/finesse/api/Queue/1004</uri>`
	`         </Queue>`
	`      </data>`
	`</Update>`

Notification Triggers:	Finesse publishes a notification • every 10 seconds, if queue statistics change • when a queue name changes • when a queue is deleted

User/Queue Notification

Finesse sends a User/Queues notification when a queue is added or removed from the user's list of queues or if a queue assigned to that user is removed from the system.

Note Finesse sends notifications for this node only for a stand-alone Finesse deployment with Unified CCE. Notifications for this node are not sent for a coresident Finesse deployment with Unified CCX.

Format:	XML
Node:	/finesse/api/User/{id}/Queues
Source:	/finesse/api/User/{id}/Queues
Data:	User/Queues object

Payload (POST):	
	```
	<Update>
	    <event>{post}</event>
	    <source>/finesse/api/User/{id}/Queues</source>
	    <requestId></requestId>
	    <data>
	       <Queues>
	          <Queue>
	             <uri>/finesse/api/Queue/{id}</uri>
	             <name>Sales</name>
	             <statistics>
	                 <callsInQueue>3</callsInQueue>

	<startTimeOfLongestCallInQueue>2012-02-15T17:58:21Z</startTimeOfLongestCallInQueue>

	                 <agentsReady>1</agentsReady>
	                 <agentsNotReady>2</agentsNotReady>
	                 <agentsTalkingInbound>3</agentsTalkingInbound>
	                 <agentsTalkingOutbound>4</agentsTalkingOutbound>
	                 <agentsTalkingInternal>5</agentsTalkingInternal>
	                 <agentsWrapUpNotReady>6</agentsWrapUpNotReady>
	                 <agentsWrapUpReady>7</agentsWrapUpReady>
	             </statistics>
	           </Queue>
	          ... more queues ...
	      </Queues>
	    </data>
	</Update>
	```

Payload (DELETE):	
	```
	<Update>
	    <event>{delete}</event>
	    <source>/finesse/api/User/{id}/Queues</source>
	    <requestId></requestId>
	    <data>
	       <Queues>
	          <Queue>
	             <uri>/finesse/api/Queue/{id}</uri>
	          </Queue>
	          <Queue>
	             <uri>/finesse/api/Queue/{id}</uri>
	          </Queue>
	          <Queue>
	             <uri>/finesse/api/Queue/{id}</uri>
	          </Queue>
	          ... more queues ...
	      </Queues>
	    </data>
	</Update>
	```

Sample Notification Payload (POST):	
	```
	Update>
	    <event>post</event>
	    <source>/finesse/api/User/1001001/Queues</source>
	    <requestId></requestId>
	    <data>
	       <Queues>
	          <Queue>
	             <uri>/finesse/api/Queue/1215</uri>
	             <name>Sales</name>
	             <statistics>
	                 <callsInQueue>3</callsInQueue>

	<startTimeOfLongestCallInQueue>2012-02-15T17:58:21Z</startTimeOfLongestCallInQueue>

	                 <agentsReady>1</agentsReady>
	                 <agentsNotReady>2</agentsNotReady>
	                 <agentsTalkingInbound>3</agentsTalkingInbound>
	                 <agentsTalkingOutbound>4</agentsTalkingOutbound>
	                 <agentsTalkingInternal>5</agentsTalkingInternal>
	                 <agentsWrapUpNotReady>6</agentsWrapUpNotReady>
	                 <agentsWrapUpReady>7</agentsWrapUpReady>
	             </statistics>
	           </Queue>
	          ... more queues ...
	      </Queues>
	    </data>
	</Update>
	```

Sample Notification Payload (DELETE):	
	```
	<Update>
	    <event>delete</event>
	    <source>/finesse/api/User/1001001/Queues</source>
	    <requestId></requestId>
	    <data>
	       <Queues>
	          <Queue>
	             <uri>/finesse/api/Queue/1326</uri>
	          </Queue>
	          <Queue>
	             <uri>/finesse/api/Queue/1364</uri>
	          </Queue>
	          <Queue>
	             <uri>/finesse/api/Queue/1389</uri>
	          </Queue>
	          ... more queues ...
	      </Queues>
	    </data>
	</Update>
	```

Notification Triggers:	• A queue is added or removed from the user's list of queues. • A queue assigned to the user is removed from the system.

Media Notification

Finesse sends a Media notification when information about a user in a Media Routing Domain changes.

Format:	XML
Node:	/finesse/api/User/{id}/Media
Source:	/finesse/api/User/{id}/Media/{mrdId}

Data:	Media

Payload:	`<Update>`
	`   <event>{put|delete}</event>`
	`   <source>/finesse/api/User/{id}/Media/{mrdId}</source>`
	`   <data>`
	`      <Media>`
	`      <!-- full Media object -->`
	`      </user>`
	`   </data>`
	`</Update>`

Sample Notification Payload:	`<Update>`
	`    <event>put</event>`
	`    <source>/finesse/api/User/1001004/Media/5000</source>`
	`    <requestId>xxxx-xxxx</requestId>`
	`    <data>`
	`        <Media>`
	`            <uri>/finesse/api/User/1001004/Media/5000</uri>`
	`            <description>Chat MRD</description>`
	`            <dialogLogoutAction>CLOSE</dialogLogoutAction>`
	`            <id>5000</id>`
	`            <interruptible>true</interruptible>`
	`            <maxDialogLimit>10</maxDialogLimit>`
	`            <name>Cisco_Chat_MRD</name>`
	`            <ReasonCode>`
	`                <category>NOT_READY</category`
	`                <code>10</code>`
	`                <forAll>true</forAll>`
	`                <id>16</id>`
	`                <label>Team Meeting</label>`
	`                <uri>/finesse/api/ReasonCode/16</uri>`
	`            </ReasonCode>`
	`            <reasonCodeId>16</reasonCodeId>`
	`            <routable>true</routable>`
	`            <state>NOT_READY</state>`
	`            <stateChangeTime>2015-09-11T06:55:14.782Z</stateChangeTime>`
	`        </Media>`
	`    </data>`
	`</Update>`

Notification Triggers:	• State change

Media and Dialogs/Media Asynchronous Error Notification

If an operations performed on a Media or nonvoice Dialog results in an asynchronous error, the error notifications include the error type, error code, and error constant. The ErrorMedia parameter indicates the Media Routing Domain to which the error applies.

Format:	XML
Node:	/finesse/api/User/{id}/Media /finesse/api/User/{id}/Dialogs/Media

Source:	/finesse/api/User/{id}/Media/{mrdId} /finesse/api/User/{id}/ Media/{mrdId}/Dialogs (when a Dialog is added or removed from the Dialog collection for the user, for example offered or closed.) /finesse/api/Dialog/{id} (when a Dialog within the Dialogs collection for the user is modified, for example accepted, started, paused, or wrapped up.)
Data:	Media Dialog

Payload:	`<Update>`
	`  <data>`
	`    <apiErrors>`
	`      <apiError>`
	`        <errorData>[Error Code]</errorData>`
	`        <errorMedia>5001</errorMedia>`
	`        <errorMessage>[Error Constant]</errorMessage>`
	`        <errorType>[Error Type]</errorType>`
	`      </apiError>`
	`    </apiErrors>`
	`  </data>`
	`  <event>PUT</event>`
	`  <requestId>xxx-xxxx</requestId>`
	`  <source>/finesse/api/User/{id}/Media/{mrdId}</source>`
	`</Update>`

Sample Notification Payload:	`<Update>`
	`  <data>`
	`    <apiErrors>`
	`      <apiError>`
	`        <errorData>1</errorData>`
	`        <errorMedia>5001</errorMedia>`
	
	`<errorMessage>E_ARM_STAT_AGENT_ALREADY_LOGGED_IN</errorMessage>`
	`        <errorType>Agent already logged into MRD</errorType>`
	`      </apiError>`
	`    </apiErrors>`
	`  </data>`
	`  <event>PUT</event>`
	`  <requestId>xxx-xxxx</requestId>`
	`  <source>/finesse/api/User/1001001/Media/5001</source>`
	`</Update>`

Notification Triggers:	The notification system returns this error if an operation on a Media or nonvoice Dialog results in an asynchronous error.

Media and Dialogs/Media Error Code Descriptions

Errors for Agent State and Mode Changes

Common Agent State and Mode Change Errors

This table describes common errors returned if agent state or mode changes fail.

Error Constant	Error Code	Description
E_ARM_STAT_AGENT_NOT_FOUND	2	The specified agent is not configured in CCE.
E_ARM_STAT_MRD_LIST_ENTRY_NOT_FOUND	3	The specified Media Routing Domain is not configured in CCE.
E_ARM_STAT_AGENT_NOT_LOGGED_IN	6	The specified agent is not logged into the MRD. This error is not returned when logging the agent into an MRD.

Agent Login Errors

Error Constant	Error Code	Description
E_ARM_STAT_AGENT_ALREADY_LOGGED_IN	1	The specified agent is already logged in to this MRD.
E_ ARM_STAT_CANT_LOGIN_TO_VOICE_MRD	11	The agent cannot log in to the voice MRD. The application attempted to log an agent into the voice MRD using the Media API instead of the User API.
E_ARM_STAT_LOGIN_NOT_ALLOWED_FOR_APP_PATH	27	The MRD and peripheral specified in the agent login request are not members of the application path associated with the Finesse server that sent the request.
E_ARM_STAT_PERFORMANCE_LIMIT_EXCEEDED	34	This code is used in the Packaged CCE deployment. When the PG reaches the Maximum Concurrent Number of Logged in Agents for that peripheral, all the ARMMediaLoginReqs for that Peripheral are rejected with this status code.
E_ARM_STAT_CC_OFFLINE	36	The log in request failed because the Central Controller is offline.
E_ ARM_STAT_LOGIN_TIMEOUT	37	The log in request timed out.
E_ARM_STAT_PQ_LOGIN_FAILED	38	The agent log in request to the precision queue failed.
E_ARM_STAT_LOGIN_REQUEST_ALREADY_PENDING	41	There is already a pending request for the agent to log in to the Media Routing Domain.

Agent Not Ready Errors

Error Constant	Error Code	Description
E_ARM_STAT_ALREADY_HAVE_PENDING_MAKE_AGENT_NOT_READY	9	There is already a pending request to make this agent Not Ready in this Media Routing Domain.
E_ARM_STAT_DO_THIS_WITH_TASK_SENT_RECENTLY	14	The agent cannot be made Not Ready because the agent has a pending incoming task; Finesse has received an offered dialog for the agent.
E_ARM_STAT_ALREADY_IN_REQUESTED_AGENT_STATE	39	The specified agent is already in the Not Ready state. If reason codes are enabled, then an agent state change from Not Ready to Not Ready with a different reason code is allowed.

Agent Mode Change Errors

Error Constant	Error Code	Description
E_ARM_STAT_ALREADY_HAVE_PENDING_MAKE_AGENT_NOT_ROUTABLE	8	There is already a pending request to make this agent Not Routable in this Media Routing Domain.
E_ARM_STAT_ALREADY_IN_REQUESTED_AGENT_MODE	40	The agent is already in the requested mode.

Internal Errors

If you receive these errors, Contact Cisco Technical Support for assistance.

Error Constant	Error Code
E_ARM_STAT_NO_ACTIVE_SKILL_GROUPS_IN_MRD_LIST_ENTRY	5

Errors for Dialogs

Common Dialog Errors

This table describes common errors returned if Dialog actions fail.

Error Constant	Error Code	Description
E_ARM_STAT_AGENT_NOT_FOUND	2	The specified agent is not configured in CCE.
E_ARM_STAT_MRD_LIST_ENTRY_NOT_FOUND	3	The specified Media Routing Domain is not configured in CCE.
E_ARM_STAT_AGENT_NOT_LOGGED_IN	6	The specified agent is not logged into the MRD.

Error Constant	Error Code	Description
E_ARM_STAT_TASK_OBJECT_NOT_FOUND	18	The specified dialog cannot be found.
E_ARM_STAT_INCONSISTENT_MEDIA_ROUTING_DOMAIN_IDS	20	The Media Routing Domain ID does not match the MRD ID for this skill, service, or dialog.
E_ARM_STAT_NOT_VALID_AFTER_INTERRUPT_ADVISORY_ACCEPT	30	The dialog has been interrupted by a dialog in a different MRD. Typically, this code indicates that a voice call interrupted the agent working on a chat or an email.
INVALID_DIALOG_ID: <DIALOG ID>	6030	The dialog API request is made and the synchronous response received but the dialog is removed before contacting CCE.

Internal Errors

If you receive these errors, Contact Cisco Technical Support for assistance.

Error Constant	Error Code
E_ARM_STAT_INVALID_MESSAGE_SEQUENCE	19
E_ARM_STAT_NO_OFFER_OR_PRE_CALL_RECEIVED	21
E_ARM_STAT_INCONSISTENT_AGENT_IDS	22
E_ARM_STAT_SKILL_GROUP_NOT_FOUND	32
E_ARM_STAT_SERVICE_NOT_FOUND	33

Notification Parameters

Name	Data Type	Description	Possible Values
Data	Object	Provides the new representation of the modified User, Team, Dialog, Queue, User/Queues, or Media object. This information is not provided when a user is deleted. For a Dialog or Media Error notification, provides the list of ApiError objects that represent the failure conditions detected by the server.	The entire User, Team, Dialog, Queue, or Media object in its most current, updated form. The Team object includes all of its agents. For the User/Queues object, specifies a list of queues that were added or deleted from the user's list.

Name	Data Type	Description	Possible Values
Event	String	The type of modification that occurred to the User, Team, Dialog, Queue, User/Queues, or Media object.	**PUT:** A property of the User, Dialog, Team, Queue, or Media object that was modified. **DELETE:** A User, Dialog, Team, Queue, or Media object has been deleted. For a User/Queues modification, the queues removed from the user's list of queues. **POST:** A User, Dialog, Team, Queue, or Media object has been added. For a User/Queues modification, specifies the queues that were added to the user's list of queues.

Source	String	The resource location for the User, Dialog, Team, Queue, User/Queues, or Media object that was modified.	/finesse/api/User/{id}
			/finesse/api/Dialog/{id}
			/finesse/api/Team/{id}
			/finesse/api/User/{id}/Dialogs
			/finesse/api/User/{id}/Dialogs/Media
			/finesse/api/Queue/{id}
			/finesse/api/User/{id}/Queues
			/finesse/api/User/{id}/Media

RequestId	String	The requestId that was returned when the triggering REST API request was made. If the event was unsolicited, this tag is empty. This tag is empty for a User/Queues notification.	An opaque, unique string, used to correlate the originating request with the resulting event

Managing Notifications in Third-Party Applications

For applications that aren't gadgets in the Cisco Finesse Desktop or in a third-party OpenSocial container, use one of the following methods to establish a connection with the Cisco Finesse Notification Service to subscribe to XMPP events:

- Any client-side XMPP library that supports WebSockets such as Strophe.js using the port 8445.
- Cisco Finesse Desktop EventTunnel (for browser applications only)
- XMPP over TCP based on Smack over port 5222 or 5223 (TLS)

Finesse uses the following base XMPP features:

1. session establishment

2. presence
3. roster management

   These are supported over BOSH (http-bind)/WebSocket/smack protocols.

   In addition, the only XMPP extension feature supported is (XEP-0060) Pubsub.XMPP extensions natively supported by Openfire. For example, (XEP – 0198) Stream management, (XEP-0163) PEP, (XEP-0256) Last Activity, aren't used by Finesse and wherever possible are disabled. Custom clients should ensure that only supported features are used when interacting with OpenFire.

Note

- Finesse by default uses WebSocket to connect to Cisco Finesse Notification Service. For better performance, third-party XMPP clients should connect to the Cisco Finesse Notification Service over WebSocket.
- For all types of connection methods, Cisco Finesse Notification Service expects XMPP client ping every 20 seconds.

This section describes how to use the Cisco Finesse Desktop `EventTunnel` method. This method requires knowledge of how to use `postMessage` to pass messages between different frames in the browser.

The EventTunnel.js file is located at `https://<hostname>:<port>/tunnel/EventTunnel.js` (the hostname is of the Cisco Finesse server and the port is 8445 or 7443 for HTTPS). This class is designed to be loaded within an **iframe**. This class loads in the browser application and uses `postMessage` to communicate between frames.

Access BOSH and WebSockets as follows:

BOSH: `https://<hostname>:<port>/http-bind`

WebSocket: `ws(s)://<hostname>:<port>/ws`

Note Cisco Finesse, Release 12.5(1) onward, the 7071 port (BOSH/WebSocket for HTTP) is disabled by default. Use the **utils finesse set_property webservices enableInsecureOpenfirePort true** command to enable this port.

For more information, see the *Service Properties* section in the *Cisco Finesse Administration Guide*.

Using the EventTunnel, the application can perform the following operations:

- Establish the XMPP connection
- Subscribe to XMPP nodes
- Unsubscribe from XMPP nodes

The following is a sample file that you can use to instantiate and initialize the EventTunnel in the iframe:

```
<!DOCTYPE HTML>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <script type="text/javascript">
    //Set the JabberWerx connect to unsecure because the custom authentication
    //on the XMPP server does not support encrypted credentials.
    var jabberwerx_config = {unsecureAllowed: true};
    <script type="text/javascript">
//Set the JabberWerx connect to unsecure because the custom authentication
//on the XMPP server does not support encrypted credentials.
var jabberwerx_config = {unsecureAllowed: true};
</script>

<script type="text/javascript" src="thirdparty/jquery/jquery-1.5.min.js"></script>
<script type="text/javascript" src="thirdparty/strophe/strophe.js"></script>
<script type="text/javascript" src="thirdparty/strophe/strophe.pubsub.min.js"></script>
<script type="text/javascript" src="thirdparty/util/converter.js"></script>
<script type="text/javascript" src="EventTunnel.js"></script>
<script type="text/javascript">
jQuery(document).ready(function () {
    var tunnel = new finesse.EventTunnel();
    tunnel.init();
});
</script>
</head>
</html>
```

Connect to XMPP over HTTP (BOSH/WebSocket) using Finesse EventTunnel

To initialize the XMPP connection, the following information must be passed to the EventTunnel before it can proceed:

1. Agent ID
2. XMPP Domain
3. Agent Password
4. XMPP PubSub Domain
5. Agent XMPP Resource (Optional)

The postMessage payload has the following message structure:

```
message = type + "|" + message;
```

where type is a number that has the following mapping:

Message Type	Value	Description
EVENT	0	XMPP events received by the EventTunnel and published out to the parent frame
ID	1	Agent XMPP ID
PASSWORD	2	Agent XMPP password
RESOURCEID	3	Agent XMPP resource
STATUS	4	Status of the XMPP connection published by the EventTunnel
XMPPDOMAIN	5	Domain of the XMPP service
PUBSUBDOMAIN	6	PubSub domain of the XMPP service
SUBSCRIBE	7	Request to subscribe to an XMPP node

Message Type	Value	Description
UNSUBSCRIBE	8	Request to unsubscribe form an XMPP node
PRESENCE	9	Request to subscribe to XMPP presence
DISCONNECT_REQ	11	Request to disconnect the XMPP connection. This request attempts to unsubscribe the application from all nodes to which it subscribed during the session and then disconnects the session.

For example, a postMessage call to send the agent ID is as follows:

```
message = "1|1001001"                              // 1 - type: ID, 1001001 - agent ID
tunnelFrame.postMessage(message, tunnelOrigin);    // where tunnelFrame is the frame
                                                   // corresponding to the iframe hosting
                                                   // the EventTunnel and tunnelOrigin is
                                                   // the URL of the EventTunnel i.e.
                                                   // http://<host>:<port> where host is
                                                   // the host of the Cisco Finesse
                                                   // server and port is the port of
                                                   // the Cisco Finesse Notification
                                                   // Service, either 7071 for http or
                                                   // 7443 for https
```

Be sure to also wire up a callback to receive messages using postMessage from the EventTunnel frame, for example:

```
if (window.addEventListener) { //Firefox, Opera, Chrome, Safari
    window.addEventListener("message", cb, false);
} else { //Internet Explorer
    window.attachEvent("onmessage", cb);
}
```

where cb is the callback that handles any messages received using postMessage and that can parse the messages sent by the EventTunnel.

Connect to XMPP over TCP

Any third party XMPP client can connect to the Finesse Notification Service through TCP sockets for sending and receiving notifications. You can connect to ports 5222 (non-secure connection) and 5223 (secure connection).

Note Cisco Finesse, Release 12.5(1) onward, the 5222 port (non-secure connection) is disabled by default. Set the **utils finesse set_property webservices enableInsecureOpenfirePort** to *true* to enable this port.

For more information, see *Service Properties* section in *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

Connect to Secure Port 5223 over SSL/TLS

Third party clients need to add the Finesse Notification certificate to their respective trust stores. Finesse Notification Service shares the same certificate with Cisco Finesse Tomcat. To download the certificate:

1. Sign in to the Cisco Unified Operating System Administration through the URL (https://FQDN:8443/cmplatform, where FQDN is the fully qualified domain name of the primary Finesse server and 8443 is the port number).

2. Click **Security** > **Certificate Management**.
3. Click **Find** to get the list of all the certificates.
4. In the Certificate List screen, choose **Certificate** from the **Find Certificate List where** drop-down menu, enter **tomcat** in the **begins with** option and click **Find**.
5. Click the FQDN link which appears in the **Common Name** column parallel to the listed tomcat certificate.
6. In the pop-up that appears, click the option **Download .PEM File** to save the file on your desktop.

CHAPTER 7

Finesse High Availability

Failure Scenarios

The following table lists possible failure scenarios and describes how a client can determine when a failure occurs.

Scenario	Notification mechanism
Cisco Finesse Notification Service goes down. **Note** In a Unified CCX deployment, this service is called the Cisco Unified CCX Notification Service.	Client loses XMPP connection to the Cisco Finesse Notification Service. **Note** This condition can occur while the Cisco Finesse Notification Services is running if the client loses network connectivity to the server (for example, a client experiences a complete loss of network connectivity).
Cisco Finesse Tomcat goes down.	The 'finesse' user presence becomes UNAVAILABLE (if desktop is still connected to the Cisco Finesse Notification Service).
Finesse web app goes down.	The 'finesse' user presence becomes UNAVAILABLE (if desktop is still connected to the Cisco Finesse Notification Service).
Finesse loses connection to the CTI server.	Finesse sends a SystemInfo notification of status OUT_OF_SERVICE (if desktop is still connected to the Cisco Finesse Notification Service).

Recovery

When any of the preceding failure scenarios are detected, the course of action is to attempt or detect recovery of the server on which the scenario occurred, as well as to check for the availability of an alternate server using the following criteria (when applicable):

1. The XMPP connection is down.

Periodically check the SystemInfo object for IN_SERVICE status. After the system is IN_SERVICE, attempt to re-establish the XMPP connection.

2. If desktop is still connected and a SystemInfo OUT_OF_SERVICE notification is received:

   As long as the XMPP connection remains available, wait for a SystemInfo notification that the system is IN_SERVICE.

3. A 'finesse' user UNAVAILABLE presence is received.

   As long as the XMPP connection remains available, wait for an AVAILABLE presence notification for the 'finesse' user. Then wait for the SystemInfo IN_SERVICE notification.

Desktop Presence and Forced Logout

The Finesse server subscribes to the presence of the XMPP users of the Finesse desktop to monitor the health of the connection between the server and desktop.

Under certain conditions, Finesse sends a forced logout with a reason code of 255 to the CTI server.

In a Unified CCE deployment, the actual behavior of the desktop under these conditions depends on the setting for Logout on Agent Disconnect (LOAD).

In a Unified CCX deployment, the agent is logged out.

Note Finesse takes up to 120 seconds to detect when an agent closes the browser or the browser crashes and Finesse waits 60 seconds before sending a forced logout request to the CTI server. Under these conditions, Finesse can take up to 180 seconds to sign out the agent.

The following table lists the conditions under which Finesse sends a forced logout to the CTI server:

Scenario	Desktop Behavior	Server Action	Race Conditions
The client closes, the browser crashes, or the agent clicks the Back button on the browser.	When you close the browser or navigate away from the Finesse desktop, the Finesse desktop makes a best-effort attempt to notify the server.	Finesse receives a presence notification of *Unavailable* from the client. Finesse waits 60 seconds, and then sends a forced logout request to the CTI server.	1. The agent closes the browser window. Finesse receives a presence notification of *Unavailable* for the user. Finesse tries to sign the agent out; however, that agent is already signed out. 2. If the browser crashes, it can take the Finesse server up to 120 seconds to detect that the client is gone and send a presence notification to Finesse. A situation can occur where the client signs in to the secondary Finesse server before the primary Finesse server receives the presence notification caused by the browser crash. In this case, the agent may be signed out or put into Not Ready state on the secondary Finesse server.

			3. If the Finesse desktop is running over a slower network connection, Finesse may not always receive an *Unavailable* presence notification from the client browser. In this situation, the behavior mimics a browser crash, as described in the preceding condition.
The client refreshes the browser	—	Finesse receives a presence notification of *Unavailable* from the client. Finesse waits 60 seconds before sending a forced logout request to the CTI server to allow the browser to reconnect after the refresh.	—
The client encounters a network glitch (Finesse is in service)	Because the connection to the Finesse server temporarily goes down, the client fails over to the secondary Finesse server.	The primary Finesse server receives a presence notification of *Unavailable* from the client. Because Finesse is in service, it sends a forced logout request to the CTI server for the agent.	A situation can occur where the forced logout does not happen before the client signs in to the secondary Finesse server. If the agent is on a call, the primary Finesse server sends the forced logout request after the call ends. In a Unified CCE deployment, the agent is signed out or put into Not Ready state when the call ends, even though the client is already signed in to the secondary Finesse server. In a Unified CCX deployment, the agent is signed out.
In a Unified CCE deployment, when Refresh Token has expired	Finesse desktop sends a forced logout request to the CTI server.	The Finesse server forwards the forced logout request to the CTI server.	**Load parameter = 0** • When the agent's current state is Not Ready, Ready or Wrap-Up, the agent's state after force logout is changed to Not Ready – Force Not Ready. • When the agent's current state is Talking, the Agent goes into Not-Ready – Force Not Ready state after the call ends. **Load parameter = 1** • When the agent's current state is Not Ready, Ready or Wrap-Up, the agent goes to Logged Out – System Failure. • When the agent's current state is Talking, the Agent goes to Logged Out – System Failure immediately even though the call is still active.

Failure Handling for Task Routing Clients

Task Routing applications that use the Finesse APIs must be able to handle failure scenarios involving Finesse and CCE services.

To recover REST and XMPP connections, follow the steps described for failure recovery earlier in this chapter.

Once you recover the connections, perform more actions to recover nonvoice media state and nonvoice dialogs. The actions you perform depend on whether your application is built with the Finesse REST APIs or the finesse.min.js javascript library.

Recovery Actions for Finesse REST APIs

If your application is built with Finesse REST APIs, perform these actions to recover nonvoice media state and nonvoice dialogs:

- Use the Media GET API to synchronize your application with the state of the agent in the application's media. For example: `https://finesse_server/finesse/api/User/userId/Media/mediaId`.
- If the maxDialogLimit, interruptAction, or dialogLogoutAction settings do not match the settings set by your application at sign-in time, use the Media Sign In API to reset the settings. The Sign In API returns an "agent already logged in" error. This error is expected. The API call does not affect the agent's state in the media. The call does, however, reset the agent's maxDialogLimit, interruptAction, and dialogLogoutAction settings in the media.
- Use the nonvoice Dialog LIST method to synchronize the application with the set of dialogs that the agent currently is assigned. For example: `https://finesse_server/finesse/api/User/userId/Media/ mediaId/Dialogs`.

  Typically, the set of dialogs does not change when you use this command. However, in some failure cases, such as double PG failures, the set of dialogs changes when you use this method.

Recovery Actions for Finesse.min.js Javascript Library

Media settings (maxDialogLimit, interruptAction, and dialogLogoutAction) can become out of sync after a failure.

If your application is built with finesse.min.js, when getting the media object for the application, tell the media object the media options. The finesse.min.js library uses these settings to ensure that the media object associated with your application's agent has the correct settings after recovering from a failure.

For example:

```
media = _mediaList.getMedia({
    id: mrdID,
    onLoad: handleMediaLoad,
    onError: handleMediaError,
    onChange: handleMediaChange,
    mediaOptions: {
        maxDialogLimit: 3,
        interruptAction: "IGNORE",
        dialogLogoutAction: "CLOSE"
    }
});
```

CHAPTER 8

Finesse Desktop Gadget Development

Finesse Gadgets

Gadgets are web-based software components based on HTML, CSS, and JavaScript. They allow developers to write web applications that work anywhere on the web without modification. They are defined using a declarative XML syntax that is processed by a gadget server into a format that allows them to be embedded into the following contexts:

- standalone web pages
- web applications
- other gadgets

Note Do not use the following JavaScript methods as they block the Finesse agent desktop until the pop up is dismissed. The Finesse backend process can also be interrupted by these methods which may lead to unexpected behavior.

- window.alert()
- window.prompt()
- window.confirm()
- window.showModalDialog()

Prerequisites to Develop Gadgets

For Finesse Gadget development, a basic understanding of the following is necessary:

- How web applications work
- XML
- HTML
- JavaScript

Gadget Description

The gadgets API consists of simple building blocks:

XML: is a general purpose markup language. It describes structured data in a way that both humans and computers can read and write.

XML is the language used to write gadget specifications. A gadget is an XML file, placed on the internet where Google can find it. The XML file that specifies a gadget contains instructions on how to process and render the gadget. The XML file contains all data and code for the gadget, or it can have references (URLs) on where to find the rest of the elements.

HTML: is the markup language used to format pages on the internet. The static content of a gadget is written in HTML. HTML looks similar to XML, but is used to format web documents rather than to describe structured data.

JavaScript: is a scripting language used to add dynamic behavior to your gadgets.

Gadget XML

A gadget and its XML are synonymous. The gadget XML contains all information needed to identify and render a web application. The XML gadget specification consists of the following:

Content

The **<Content>** section specifies the programming logic and the HTML elements that determine the appearance of the gadget. It defines the type of content, and either holds the content itself or has a link to external content. The gadget attributes and user preferences are combined with programming logic and formatting information to become a running gadget.

<Content> provides the actual HTML, CSS, and JavaScript to be rendered by the gadget. Code is provided directly in the gadget XML content section for rendering and control flow. The code is processed by a gadget server and rendered in an IFRAME.

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
    <ModulePrefs title="Sample Gadget"
      …
</ModulePrefs>
    <UserPref name="scheme" display_name="scheme" default_value="" datatype ="hidden"/>
    <UserPref name="host" display_name="host" default_value="" datatype ="hidden"/>
   <UserPref name="hostPort" display_name="hostPort" default_value="" datatype ="hidden"/>


    <Content type="html">
        <![CDATA[
            <!DOCTYPE html>
            <!-- Styling -->
            <link rel="stylesheet" href="SampleGadget_Final.css" type="text/css" />
            …
            …
            <script type="text/javascript">
            …
            </script>
        ]]>
     </Content>
</Module>
```

User Preferences

The **<UserPrefs>** section allows you to pass custom properties to the gadget from the gadget XML. The custom properties have to be suffixed with the datatype attribute as hidden.

For example, `<UserPref name="myname" display_name="Name" required="true" datatype="hidden" />`.

The user preferences are defined in the XML specifications as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
    <ModulePrefs title="Sample Gadget"
      …
</ModulePrefs>
    <UserPref name="scheme" display_name="scheme" default_value="" datatype ="hidden"/>
    <UserPref name="host" display_name="host" default_value=""  datatype ="hidden"/>
   <UserPref name="hostPort" display_name="hostPort" default_value=""  datatype ="hidden"/>


    <Content type="html">
        <![CDATA[
            <!DOCTYPE html>
            <!-- Styling -->
            <link rel="stylesheet" href="SampleGadget_Final.css" type="text/css" />
<!-- Finesse Library -->
 <script type="text/javascript"
src="__UP_scheme__://__UP_host__:__UP_hostPort__/desktop/assets/js/finesse.min.js"></script>

            …
            …
            <script type="text/javascript">
            …
            </script>
        ]]>
    </Content>
</Module>
```

Note that for each User Preference, “hangman variables” can be substituted into supported gadget specification fields. Hangman variables are of the form **__<TYPE>_<ID>__**, and are replaced with string values. For each provided User Pref with key foo and value bar, hangman expansion **__UP_foo__ = bar**. Hence, in the above code user preference scheme is available as **__UP_scheme__**. Similarly, for other User Preferences the hangman variables are dynamically substituted. Also, as the datatype value is specified as hidden, the user preferences pop up for the agent to enter their own data does not show up on the gadget.

User preferences are accessed from your gadget using the user preferences JavaScript API, for example:

```
<script type="text/javascript">
  var prefs = new gadgets.Prefs();
  var someStringPref = prefs.getString("StringPrefName");
  var someIntPref = prefs.getInt("IntPrefName");
  var someBoolPref = prefs.getBool("BoolPrefName");
</script>
```

Gadget JavaScript

Contains the business logic for the gadget. It can be written inside the gadget XML under the content section or an external JavaScript file can be created which can then be referred to using the src attribute in the **<script>** tag.

Gadget CSS

Contains the complete styling of the gadget. Similar to the JavaScript, CSS can also be referred to as an external file using href attribute in **<link>** tag.

Gadget Behavior

Rendering a gadget at the page level removes the title bar from the gadget layout.

Components

Components are simple scripts that are loaded into the desktop directly at predefined positions as directed by the layout, without an enclosing frame and its document.

Components have been introduced in the desktop to overcome a few rendering limitations and performance considerations inherent to gadgets.

Components are listed in the desktop layout using the <component> tag. Currently, the layout validations prevent custom components from being created. Hence, only default components are allowed in the desktop layouts. The default desktop functionalities are currently registered as components to provide flexibility and to reduce the load on the server.

Simple Example Gadget

Do the following to create and deploy a gadget:

- Use any text editor to write your gadget specification.
- Host the gadget on any web server. See Enable or Reset 3rdpartygadget Account, on page 449.
- Add the gadget to the Finesse Container which can run gadgets. See Upload Third-Party Gadgets, on page 450.

Example Gadget

Use the following lines of code to build a simple gadget. This gadget displays the message "Hello, world!". Copy the following lines of code into a new file named hello_world.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
  <ModulePrefs title="hello world example" />
  <Content type="html">
     <![CDATA[
       Hello, world!
     ]]>
  </Content>
</Module>
```

Note the following about the "Hello World" example:

- Gadgets are specified in XML. The first line is the standard way to start an XML file. This must be the first line in the file.
- The **<Module>** tag indicates that this XML file contains a gadget.
- The **<ModulePrefs>** tag contains information about the gadget such as its title, description, author, and other optional features.
- The line **<Content type="html">** indicates that the gadget's content type is HTML.
- <![CDATA[...*insert HTML here*...]]> is used to enclose HTML when a gadget's content type is html. It tells the gadget parser that the text within the CDATA section should not be treated as XML. The CDATA section typically contains HTML and JavaScript.
- **</Content>** signifies the end of the Content section.
- **</Module>** signifies the end of the gadget definition.

Note For a Finesse specific example, download the LearningSampleGadget from https://github.com/CiscoDevNet/finesse-sample-code/tree/master/LearningSampleGadget, which provides step by step instructions in learning some of the objects in the finesse.min.js library.

Note Portions of this page are reproduced from work created and shared by Google, see https://developers.google.com/terms/site-policies and used according to terms described in the Creative Commons 3.0 Attribution License, see https://creativecommons.org/licenses/by/3.0/. For more information about OpenSocial gadgets, see https://developers.google.com/gadgets/docs/overview. Note that not all OpenSocial gadget features are available in the Finesse container.

Gadget Limitations

Cisco Finesse, Release 12.5(1) allows an agent or a supervisor to drag-and-drop a gadget to the required position on the desktop layout. The drag-and-drop feature is not applicable for gadgets that do not have a defined title.

Example: Gadget without Title

```
<ModulePrefs>
    <Require feature="pubsub-2"></Require>
    <Require feature="setprefs"></Require>
    <Require feature="osapi"></Require>
    <Require feature="dynamic-height"></Require>
</ModulePrefs>
```

Example: Gadget with Title

```
<ModulePrefs title="SampleGadget"
       description="Hello">
    <Require feature="settitle"/>
    <Require feature="pubsub-2"></Require>
    <Require feature="setprefs"></Require>
    <Require feature="osapi"></Require>
    <Require feature="dynamic-height"></Require>
</ModulePrefs>
```

Import Finesse JavaScript API

For gadgets to work properly, they need to import the Finesse JavaScript library hosted on the Finesse server.

Hosting Third-Party Gadgets on Web Server

To import the JavaScript library, the Finesse FQDN needs to be provided inside the import statement. For building the finesse.min.js URL, we need to retrieve the following properties from the gadget preferences:

1. **scheme**: https
2. **hostname**: FQDN of the Finesse server
3. **port**: port of the Finesse service

These properties are inside the gadget preferences as part of Finesse container initialization. In your gadget XML:

- Define the user preferences that will be used for building the finesse.min.js import statement.

```
<UserPref name="scheme" display_name="scheme" default_value="" datatype ="hidden"/>
<UserPref name="host" display_name="host" default_value="" datatype ="hidden"/>
<UserPref name="hostPort" display_name="hostPort" default_value="" datatype ="hidden"/>
```

- Import the finesse.min.js file.

```
<script type="text/javascript"
    src="__UP_scheme__://__UP_host__:__UP_hostPort__/desktop/assets/js/finesse.min.js">
</script>
```

Hosting Third-Party Gadgets on Finesse Server

Third-party gadgets can be hosted on the Finesse server inside the 3rdpartygadget directory. See Upload Third-Party Gadgets, on page 450.

Since the third-party gadget is hosted on the Finesse server, you can import the Finesse JavaScript API with a relative URL.

```
<script type=”text/javascript”src=”/desktop/assets/js/finesse.min.js”></script>
```

alternateHosts Configuration

The <gadget> element in the Finesse Layout XML provides an attribute to specify alternate hosts from which the gadget can be loaded. This allows the Cisco Finesse desktop to load the gadget using a different host if the primary server is unavailable.

The **alternateHosts** attribute contains a comma-separated list of FQDNs that will be used if the primary-host-FQDN is unavailable.

```
<gadget alternateHosts="host1,host2,host3,...">
         https://<primary-host-FQDN>/<gadget-URL>
    </gadget>
```

The **alternateHosts** attribute is only applicable for gadgets with an absolute URL. That is URLs containing the FQDN of a host, an optional port, and the complete URL path to the gadget. For example: <gadget alternateHosts="host1,host2">*https://primary host/relative_path</gadget>*

If loading the gadget from the primary-host fails, the Cisco Finesse container attempts to load the gadget from the alternate hosts in the order specified in the **alternateHosts** attribute.

The Cisco Finesse desktop may fail to load the gadget even if some of the hosts are reachable. In such cases, refresh the Cisco Finesse desktop.

When the gadget is specified with a relative URL, for example: *<gadget >/3rdpartygadgets/relative_path</gadget>*, the **alternateHosts** attribute does not apply and is ignored by the Cisco Finesse desktop.

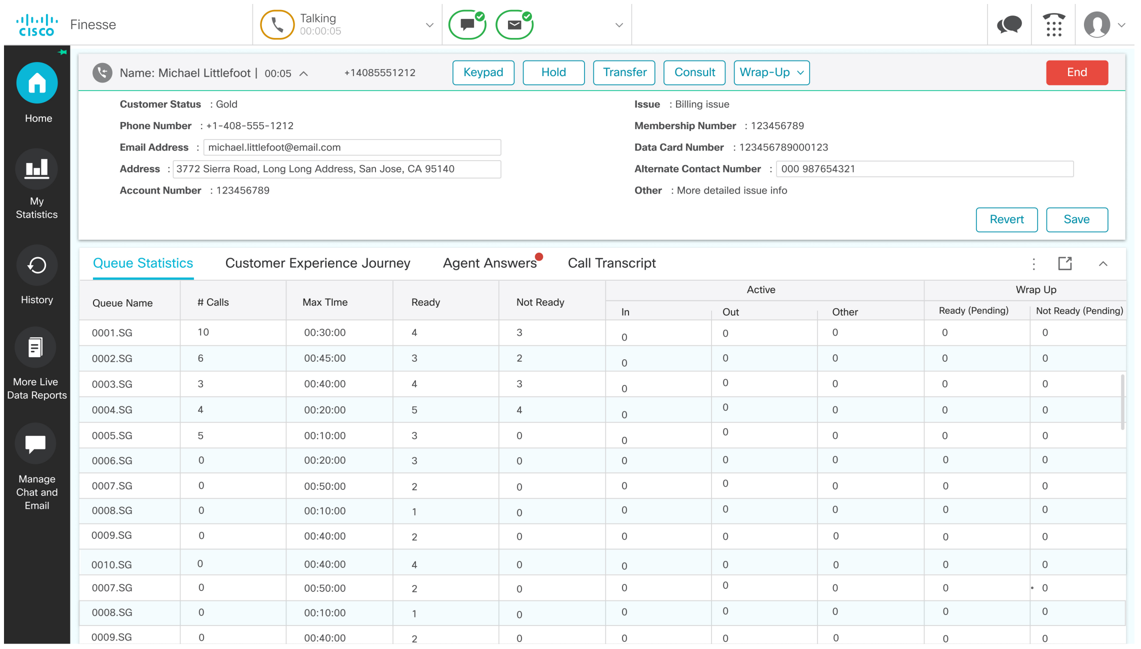
Note If the host serving the gadget fails after the Cisco Finesse desktop was successfully loaded, the desktop must be refreshed in order to load the gadget from an alternate host. The gadget does not implement its own failover mechanism.

Headless Gadget Configuration

Headless gadgets are gadgets which do not need a display space, but can be loaded and run like a background task in the browser. The **Hidden** attribute (optional) is used to support headless gadgets in the layout XML. When an attribute is set to "hidden=true", then the gadget is loaded by the container, but will not be displayed. The default value set for the attribute is "false".

Multi-Tab Gadgets

The Multi-Tab gadgets feature allows the user to view multiple gadgets in a tabular structure. The sequence of the tabs is based on the order in the desktop layout. Tabbed gadgets allow users to maximize the space on the Finesse desktop by consolidating multiple gadgets into a single space. It also allows the users to maximize or minimize the gadget as well as restore or collapse the gadget based on their viewing preference.



Name: Michael Littlefoot | 00:05 +14085551212

Customer Status	: Gold	**Issue**	: Billing issue
Phone Number	: +1-408-555-1212	**Membership Number**	: 123456789
Email Address	: michael.littlefoot@email.com	**Data Card Number**	: 123456789000123
Address	: 3772 Sierra Road, Long Long Address, San Jose, CA 95140	**Alternate Contact Number**	: 000 987654321
Account Number	: 123456789	**Other**	: More detailed issue info

Queue Statistics | Customer Experience Journey | Agent Answers | Call Transcript

Queue Name	# Calls	Max TIme	Ready	Not Ready	Active			Wrap Up	
					In	Out	Other	Ready (Pending)	Not Ready (Pending)
0001.SG	10	00:30:00	4	3	0	0	0	0	0
0002.SG	6	00:45:00	3	2	0	0	0	0	0
0003.SG	3	00:40:00	4	3	0	0	0	0	0
0004.SG	4	00:20:00	5	4	0	0	0	0	0
0005.SG	5	00:10:00	3	0	0	0	0	0	0
0006.SG	0	00:20:00	3	0	0	0	0	0	0
0007.SG	0	00:50:00	2	0	0	0	0	0	0
0008.SG	0	00:10:00	1	0	0	0	0	0	0
0009.SG	0	00:40:00	2	0	0	0	0	0	0
0010.SG	0	00:40:00	4	0	0	0	0	0	0
0007.SG	0	00:50:00	2	0	0	0	0	0	0
0008.SG	0	00:10:00	1	0	0	0	0	0	0
0009.SG	0	00:40:00	2	0	0	0	0	0	0

For more information on configuring the multi-tab gadget layout, see *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

For more information, see container services introduced as a part of this feature — hideMyGadget(), hideMyGadgetNotification(), setMyGadgetTitle(title), showMyGadget(), showMyGadgetNotification() is available at Container Services, on page 457.

.

Best Practices for Gadget Development

Each new gadget adds more load to the Cisco Finesse server and caution must be observed when adding gadgets to all users. Gadgets must comply with certain performance guidelines to allow the best possible performance, which results in faster Cisco Finesse failover.

For more information on deployment practices and guidelines to ensure optimal failover performance, see *Guidelines for Optimal Desktop Failover* and *Failover Planning* sections in *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

Note In Unified CCE, Cisco Finesse deployments with more than eight gadgets per agent for the maximum supported 2000 users, must deploy the Cisco Finesse OVA with 8 CPUs to ensure faster failover time.

The actual number of gadgets that require the Cisco Finesse OVA with 8 CPUs depends on the gadgets and the number of requests that they invoke. In Unified CCE, Cisco Finesse deployments must be monitored for CPU consumption.

General Gadget Development Guidelines

The following are the general gadget guidelines that are applicable for both gadgets uploaded in Cisco Finesse 3rd party folder and gadgets hosted in 3rd party servers.

- Use XML-based gadget URLs instead of dynamic JSP-based gadget URLs to prevent extra calls to the Finesse server.
- Bundle and pack the resources for faster downloads.
- Use finesse.min.js, which is compressed and smaller in size over finesse.js.
- Prevent loading the Finesse desktop with **`bypassServerCache=true&nocache`** as a query parameter in the desktop URL.
- Ensure static resources used by the gadget is cached by the browser.
- External gadget hosting servers must prefer CA-signed certificates for easy integration with the browser. If they are self-signed, then import those certificates into the agent browser.

  For more information, see *Accept Security Certificates* section in *Cisco Finesse Agent and Supervisor Desktop User Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-user-guide-list.html.
- Gadgets must cache configuration data retrieved after the gadget load, and reuse them after failover using `DesktopCache` JavaScript API. Use the `DesktopCache` JavaScript API to prevent calls being made to the external servers during failover.

Guidelines for Gadgets Uploaded to Finesse 3rdpartygadget Account

- Application data that are not susceptible to change across sessions can be cached in the browser using `DesktopCache` API and reused.
- Add exclusions for finesse.js and the 3rd party JavaScript files in the gadget XML.

Guidelines for Gadgets Hosted on 3rd Party Servers

- If a gadget is loaded from a 3rd party server, then make REST API calls directly to that server without proxying the requests through Shindig (avoid gadget.makeRequest() calls).
- Load the static data as scripts or HTML and not as active-content (JSP files).

Guidelines for JSP Gadgets

- Efficient failover requires converting all the JSP-based gadgets to XML-based gadgets.
- In Unified CCX Release 12.5(1), all the CUIC and LiveData gadgets are converted to XML-based gadgets.
- In Unified CCE, after upgrading CUIC to Release 12.5(1) use the CLI command **utils finesse layout updateCuicGadgetUrl** to change the JSP references of CUIC gadgets to XML with no functional changes.

  For more information, see *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

Note The webproxy in Cisco Finesse cannot cache any JSP content.

Gadget Deployment Guidelines

- Verify the new gadgets by accessing a configured layout using `nocache` query parameter in the desktop URL.
- When gadgets are successfully deployed, access the changed layout to verify the gadgets.

Supported OpenSocial Features

The Finesse Desktop supports OpenSocial Core Gadget Specification 1.1.

Gadget Specification XML Features

The following table lists supported features that can be specified in the Gadget Specification XML or are available as an API for use in the JavaScript code of a gadget.

Name	Description
Locale	The <Locale> element specifies the locales that the gadget supports. The Finesse Desktop Gadget Container takes the locale provided by the browser and renders the gadget with the specific message bundle when available.
ModulePrefs: Scrolling	The Scrolling attribute of the ModulePrefs tag renders the gadget frame with a value of auto for scrolling. When the content exceeds the viewport, the browser renders a vertical or horizontal scrollbar. For a better user experience, use the gadgets.window.adjustHeight API to dynamically resize the gadget as needed instead of using this feature.
ModulePrefs: Title	The string provided is used for the title of the gadget shown in the title bar. You can also use the gadgets.window.setTitle API to set the title at runtime, which may offer more flexibility.
loadingindicator	Displays a loading message while the gadget is loading.

Required Module pref Feature

Finesse requires that all gadgets use the following module pref feature:

<Require feature="pubsub-2" />: This feature is required for the gadget to load in the OpenAjax Hub.

Note Before you can access the authorization string through the gadget prefs, you must first import the Finesse JavaScript library.

Loading Indicator Feature

The loading indicator is an OpenSocial feature that displays a loading message over gadgets while they are loading. This feature allows you to provide a consistent user experience within Finesse.

Requesting the Loading Indicator

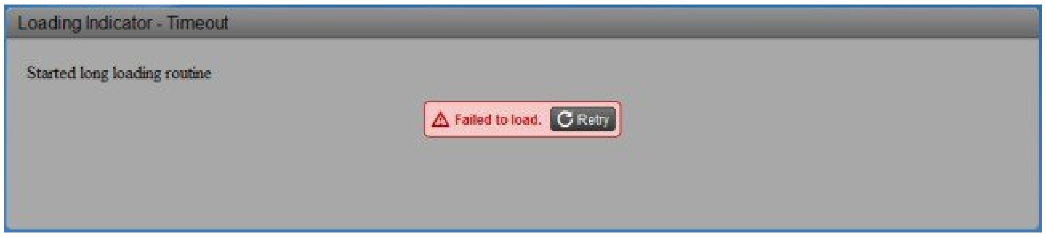
Use the following to request the loading indicator in the gadget ModulePrefs:

```
<ModulePrefs>
  <Require feature="loadingindicator">
    <Param name="manual-dismiss">false</Param>
    <Param name="loading-timeout">10</Param>
  </Require>
</ModulePrefs>
```

Parameter	Type	Description	Possible Values	Notes
loading-timeout	Integer	The number of seconds to wait before displaying the Retry button. If the loading indicator is dismissed within this time, the Retry button does not appear. Set this to a number that is appropriate for your gadget.	integers	Optional parameter. Default is 10.
manual-dismiss	Boolean	This parameter determines whether the gadget dismisses the loading indicator. If set to false, the feature code dismisses the loading indicator when the gadget has loaded. However, the indicator may be dismissed too soon because the gadget may load before all gadget initialization code is complete. To manually dismiss the loading indicator, set this parameter to true, and then configure the gadget to call gadgets.loadingindicator.dismiss() after the gadget is loaded and initialized.	true, false	Optional parameter. Default is false.

When the gadget is loading, if the loading timeout is reached, the loading indicator changes to a timeout message and displays a Retry button that the user can click to reload the gadget.

Figure 11: Loading Indicator - Timeout



You can change any of the strings displayed by the loading indicator by configuring the gadget to call the following JavaScript methods:

- gadgets.loadingindicator.updateLoadingMessage(text)
- gadgets.loadingindicator.updateTimeoutMessage(text)
- gadgets.loadingindicator.updateRetryButtonText(text)

APIs Available to Gadget JavaScript

The following table lists the available APIs and methods.

Name	Parameters	Description
<static> gadgets.window.adjustHeight(opt_height)	opt_height (integer)—Preferred height in pixels. This parameter is optional. If the opt_height is not specified, the API attempts to fit the gadget to its content.	Adjusts the height of the gadget.
<static> gadgets.window.setTitle(title)	title (string)—Preferred title of the gadget.	Sets the title of the gadget.
<static> gadgets.io.makeRequest (url, callback, opt_params)	url (string)—Address from which content is fetched. callback (function)—Run after content from the url is fetched. opt_params (Map<String, String>)—Additional optional parameters to pass to the request.	Fetches content from the provided URL and feeds that content into the callback function. **Note** The makeRequest call to the Shindig server is a POST request.
<static> gadgets.views.requestNavigateTo (view)	view (string)—The view type to which the gadget is requesting to change.	Sets the view type of the gadget. If the parameter value equals "canvas", the gadget is requesting to be maximized within the tab on which it resides. If any other value is provided, the gadget is requesting to be restored to its default view.
<static> gadgets.loadingindicator.dismiss()	None	Dismisses the loading indicator so that the message is no longer visible.
<static> gadgets.loadingindicator.showLoading()	None	Displays a loading indicator message over the gadget.
<static> gadgets.loadingindicator.showRetry()	None	Displays an error message over the gadget stating that the gadget failed to load, along with a Retry button. When the user clicks the Retry button, the container reloads the gadget.
<static> gadgets.loadingindicator.updateLoadingMessage(text)	text (string)—Text to display as the loading message.	Changes the message that appears when the gadget is loading.

Name	Parameters	Description
<static> gadgets.loadingindicator.updateTimeoutMessage(text)	text (string)—Text to display when the gadget loading has timed out.	Changes the message that appears when the gadget loading times out.
<static> gadgets.loadingindicator.updateRetryButtonText(text)	text (string)—Text to display on the Retry button.	Changes the message that appears on the Retry button.

Gadget Preferences

The gadgets.Prefs class provides access to user preferences, module dimensions, and messages. Clients can access their preferences by constructing an instance of gadgets.Prefs (and optionally, passing in their module ID). Gadget preferences can then be set using the standard OpenSocial gadget APIs.

```
var myPrefs = new gadgets.Prefs();
myPrefs.set(“counter”, count +1);
```

In the Finesse Desktop, gadget preferences persist in the browser. After a gadget sets its preferences, anytime that gadget is constructed in the same browser, these preferences continue to be available through the APIs.

```
var myPrefs = new gadgets.Prefs();
helloValue = myPrefs.getString(“hello”);
```

Note Do not use preferences to persist critical application data. This data is stored in the browser and may be manually purged by the user at will. This storage is meant for preferences (similar to the type of information that is typically stored inside a cookie), and not for complex application data. Additionally, when the browser runs out of the allocated storage space, this data is purged.

If special characters are expected in the value of the preference, they should be escaped inbound and unescaped outbound, as shown in the following example:

```
var myPrefs = new gadgets.Prefs(),
myPrefs.set("hello", gadgets.util.escapeString("!@#$%^&*()<>?");
…
var myPrefs = new gadgets.Prefs(),
helloValue = gadgets.util.unescapeString(myPrefs.getString("hello"));
```

Note Do not use special characters within the name of the preference. The use of special characters within the name of the preference is not supported.

Caveats

Although OpenSocial is a web standard, gadgets may exhibit different behaviors in various OpenSocial containers. You should always thoroughly test gadgets in Finesse to ensure that functionality is in accordance with customer requirements. The Finesse team will document known issues as they are discovered to help customers and partners build gadgets for the Finesse Desktop.

Gadget Caching

When gadget caching is enabled, the contents are cached in the Finesse server cache and Finesse gadget container. If changes are made to the code of an existing gadget, then perform one of the following:

- Restart Cisco Finesse tomcat.
- Pass a `nocache` parameter in the URL to clear the cache and use the CLI command **utils webproxy cache clear shindig** to clear the Shindig cache.

  You can pass the `nocache` parameter at the root level or at the desktop web application. For example,

  - https://server?nocache
  - https://server/desktop?nocache
  - https://server/desktop/container?nocache

Notifications on Finesse Desktop

The Finesse desktop contains support for OpenSocial Core Gadget Specification 1.1. OpenSocial Core Gadget Specification 1.1 supports an intergadget notification system that is based on the OpenAjax Hub 2.0 Specification.

The Finesse desktop automatically establishes a XMPP connection to the Notification Service upon sign-in. The Finesse desktop publishes notifications that it receives from the Notification Service to OpenAjax Hub topics. An OpenAjax topic is a string name that identifies a particular topic type to which a client can subscribe or publish. Gadgets must subscribe to these topics to receive notifications.

If the XMPP connection is disconnected, the Finesse desktop attempts to recover based on the recovery strategy. If the XMPP connection cannot be re-established, the Finesse Desktop triggers a failover to the alternate Finesse server.

Review the OpenSocial and OpenAjax Hub specifications before you implement gadget support for notifications on the Finesse Desktop.

Finesse Notifications in Third-Party Containers

Strict requirements must be followed to leverage the Finesse Desktop notification framework on a third-party container.

1. Clients must add a specific Finesse gadget, which establishes the XMPP connection and publishes notifications to Finesse-specific OpenAjax topics
2. Third-party containers (that is, those other than the Finesse Desktop) must provide support for the OpenSocial Core Gadget Specification 1.1 to ensure that gadgets can subscribe to Finesse-specific notifications through the OpenAjax Hub.

Finesse Topics

A gadget that is within the Finesse environment has the ability to subscribe or publish to a set of Finesse Desktop topics via OpenAjax Hub. The following sections provide details for the available topics.

Connection Information

Topic Name	finesse.info.connection
Topic Type	Gadgets subscribe to this topic.

Gadgets subscribe to the finesse.info.connection topic to receive status information about the XMPP connection, which is automatically established by the Finesse Desktop or a Finesse Desktop gadget (within a non-Finesse container). Connection status information can be used to determine the state of the connection so that a gadget can act appropriately. Additionally, a resource ID is provided in the published data to allow the gadget to construct a subscribe request to the Finesse Web Services. Connection information is published every time there is a connection state change.

The published data is a JavaScript object with the following properties:

```
{
    status: string,
    resourceID: string
}
```

The *status* parameter describes the XMPP connection status. It can have any one of the following values:

- connected
- connecting
- disconnected
- disconnecting
- reconnecting
- unloading

Note A XMPP connection status of "unloading" indicates that an action in the browser (such as refreshing the browser or clicking the back button) caused the XMPP connection to initiate the unloading process.

The *resourceID* parameter is a unique identifier for the XMPP connection. Although the resourceID parameter is provided with every connection status change, the ID is not available until after a XMPP connection has been successfully established. It is possible that the XMPP connection reconnects with a different resourceID.

A situation can occur where a gadget is loaded after the Finesse Desktop or gadget has already published connection information. In this case, have the gadget publish a request to a Finesse request topic, which forces the Finesse Desktop to publish the connection information again. For more information, see Finesse Requests.

Finesse Notifications

Topic Name	finesse.api.[resourceObject].[resourceID]
Topic Type	Gadgets subscribe to this topic.

If a user has any subscriptions for a particular notification, either created by the Finesse Desktop or by an explicit subscribe request (see *Subscription Management on the Finesse Desktop*), the Cisco Finesse Notification Service delivers updates through the established XMPP connection. The Finesse Desktop automatically handles the management of the XMPP event connection to the Notification Service. Any notifications that are delivered through the connection are converted to JavaScript Object, and then published by the Finesse Desktop to an OpenAjax Hub topic. The name of the topic matches the node on the Finesse Notification Service on which the notification was published. However, to comply with OpenAjax topic conventions, all slashes (/) are replaced with dots (.) and the leading slash is removed.

To receive notifications, the gadgets must

1. Subscribe to the OpenAjax topic for a particular notification feed. This action ensures that no notifications are missed after sending the subscription request to Finesse Web Services.
2. If required, make a request to the Cisco Finesse Notification Service to create a subscription for the notification feed (see *Subscription Management on the Finesse Desktop*).

When connecting to the Cisco Finesse Notification Service, you must always specify a resource to identify your connection. Issues occur if the resource is omitted when the connection is created.

The resource “desktop” is reserved for the Finesse Desktop. Do not use this resource for other connections as it causes a conflict with the Finesse Desktop.

In Finesse, each notification type has an equivalent topic to which gadgets can subscribe. For a list of available Finesse notifications, see *Cisco Finesse Notifications* and look under the "node" property. These notifications are structured as follows:

```
{
    content : Raw object payload as a String,
    object : JavaScript object representation of the payload
}
```

Finesse Requests

Topic Name	finesse.info.requests
Topic Type	Gadgets publish to this topic.

Communication between gadgets and the Finesse Desktop or other gadgets is done through inter-gadget notification via OpenAjax Hub. A gadget can send an operation request to the Finesse Desktop by publishing a request object to the Finesse request topic.

The gadget must construct an object to be published to the request topic with the following structure:

```
{
    type: string,
    data: object
}
```

The *type* parameter describes the request type.

The *data* parameter provides additional information for the Finesse Desktop to respond to the request. The contents of this data depends on the type of request.

The following sections describe the different types of requests supported.

Note More request types may be added in the future.

ConnectionInfoReq

Sending an "ConnectionInfoReq" request forces the Finesse Desktop to publish a connection information object to all gadgets subscribed to the *finesse.info.connection* topic. This request allows gadgets to determine the current state of the XMPP connection and retrieve the resource ID. The gadget must be subscribed to the connectionInfo topic to receive the event.

The gadget should publish the following object to the topic *finesse.info.requests*:

```
{
    type: "ConnectionInfoReq",
    data: { }
}
```

It is possible that the gadget may come up before the Finesse Desktop is ready to start responding to a request to send connection information. For this reason, gadgets should subscribe to the *finesse.info.connection* topic regardless. When the Finesse Desktop or gadget is ready, it starts publishing connection information immediately.

Note The topic *finesse.info.connection* is shared across all subscribed gadgets. Gadgets that subscribe to this topic may receive duplicate notifications. Gadgets must be able to handle duplicate notifications appropriately.

ConnectionReq

Sending a "ConnectionReq" forces the Finesse Desktop to attempt to establish a XMPP connection with the Notification Service. This request can only go through if either no active connection currently exists or if the current connection is in the "disconnected" state.

The gadget should publish the following object to the topic *finesse.info.requests*:

```
{
    type: "ConnectionReq",
    data: {
        id: ID,
        password: password,
        xmppDomain: xmppDomain
    },
}
```

The *id* and *password* parameters specify the ID and password of the XMPP user for which to establish an XMPP connection. The *xmppDomain* parameter specifies the domain of the XMPP server.

SubscribeNodeReq

Sending a "SubscribeNodeReq" request causes the managed XMPP connection to send an XEP-0060 standard subscribe request (described in About Cisco Finesse Notifications) to subscribe to the notification feed for the specified node. The response to this request is published on the response topic finesse.info.responses.{invokeID}, where the invokeID must be generated by the gadget to identify this unique request and subscription. For more details, see Finesse Responses. The Cisco gadgets use an RFC1422v4-compliant universally unique identifier (UUID) for this invokeID.

To guarantee that the gadget receives the response, it must subscribe to the response topic (on the OpenAjax Hub) of its self-generated invokeID before sending the following object to the topic finesse.info.requests:

```
{
    type: "SubscribeNodeReq",
    data: {
        node: "/finesse/api/Team/{id}/Users" // the node of interest
    },
    invokeID: "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx"
}
```

The *node* parameter specifies the node to subscribe to. The *invokeID* parameter is self-generated and is used to track this particular subscription. This parameter is also used as part of the OpenAjax topic to which the response of the request is published.

UnsubscribeNodeReq

Sending an "UnsubscribeNodeReq" request causes the managed XMPP connection to send an XEP-0060 standard unsubscribe request (described in section 7.1 About Cisco Finesse Notifications) to unsubscribe from the specified node. The response of this request is published on the response topic finesse.info.responses.{invokeID}, where the invokeID must be generated by the gadget to identify this unique request. For more details, see Finesse Responses. The Cisco gadgets use an RFC1422v4-compliant UUID for this invokeID. For more details, see the Finesse SDK.

To guarantee that the gadget receives the response, it must subscribe to the response topic (on the OpenAjax Hub) of its self-generated invokeID before sending the following object to the topic finesse.info.requests:

```
{
    type: "UnsubscribeNodeReq",
    data: {
        node: "/finesse/api/Team/{id}/Users",
        subid: "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    },
    invokeID: "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxy"
}
```

The *node* parameter specifies the node to subscribe to. The *subid* parameter specifies the subscription to remove, which is uniquely identified by the invokeID that was used in the subscribe request. The *invokeID* parameter is self-generated and is used as part of the OpenAjax topic to which the response of the request is published.

Finesse Responses

Topic Name	finesse.info.responses.{invokeID}
Topic Type	Gadgets subscribe to this topic.

Responses to requests are published to these channels. When a request is made, the gadget generates and specifies a unique invokeID as part of the request. This invokeID is used as the trailing token in the topic to which the response of the request is published.

Because this topic is only used to communicate the response of a single request and never used again, be sure to unsubscribe from the topic as part of the callback handler in the subscribe request. For example:

```
// Generate invokeID and construct request
var UUID = _util.generateUUID(),
data = {
    type: "ExampleReq",
    data: {},
    invokeID: UUID
},


// Subscribe to the response channel to ensure we don't miss the response
OAAsubid = gadgets.Hub.subscribe("finesse.info.responses."+ UUID, function (topic, data) {
    // Unsubscribe from the response topic to prevent memory leaks
    // Do this before processing the response in case the processing throws an exception
    gadgets.Hub.unsubscribe(OAAsubid);

    // Process the response here
});


// Publish the request after we have registered our response callback on the response topic
gadgets.Hub.publish("finesse.info.requests", data);
```

Workflow Action Event

Topic Name	finesse.containerservices.workflowActionEvent
Topic Type	Gadgets subscribe to this topic.

Gadgets subscribe to the finesse.containerservices.workflowActionEvent topic to receive workflow action events to run as a result of workflow evaluations.

Note Third-party gadgets subscribing directly to the OpenAjax Hub for the Workflow Action Event topic might cause the Finesse Workflow Engine to lose its subscription and no longer be able to run workflow actions. Third party gadgets should instead implement something like the following:

```
var _containerServices = finesse.containerservices.ContainerServices.init();
    _containerServices.addHandler("finesse.containerservices.workflowActionEvent",
function(data) {
         // Perform logic on "data", which is a WorkflowActionEvent object
        });
```

The published data is a JavaScript object with the following properties:

```
{
  uri: string,
  name: string,
  type: string,
  params: [
    {
      name: string,
      value: string,
      expandedValue: string
    }
  ],
  actionVariables: [
    {
      name: string,
      node: string,
      type: string,
      testValue: string,
      actualValue: string
    }
  ]
}
```

Field	Description
uri	In the uri, the id maps to the primary key of the WorkflowAction entry.
name	The name of the workflow action.
type	The type of workflow action. Possible value is BROWSER_POP.
params	List of Param subobjects (see below).
actionVariables	List of ActionVariable subobjects (see below). There can be at most 5 Action Variable subobjects assigned to a workflow action.

The Param subobject uses the following fields:

Field	Description
name	The name of the parameter.
value	The value of the parameter.
expandedValue	The value of the parameter with variables substituted with their values.

The ActionVariable subobject uses the following fields:

Field	Description
name	The name of the variable.
node	The XPath to extract from the dialogs XML.
type	Indicates if this is a SYSTEM or CUSTOM variable.
testValue	The value used to test the variable.
actualValue	The actual value of the variable in context of the events used by the workflow evaluation.

Finesse Container Timer

Because too many timers that run concurrently can cause issues for JavaScript, you should not use setTimeout() or setInterval() directly. The Finesse container provides a service (the TimerTickEvent) that you can leverage for your third-party gadgets.

Finesse publishes the TimerTickEvent to the OpenAJAX hub every 1000 milliseconds. To use this service:

- Have the gadget subscribe to the TimerTickEvent:

  ```
  finesse.containerservices.ContainerServices.addHandler(finesse.containerservices.ContainerServices.Topics.
  TIMER_TICK_EVENT, callback);
  ```

- Define a callback method (see boilerplate gadget tick code - _timerTickHandler()) and, optionally, an update method (see boilerplate gadget tick code - _processTick()).

Cisco provides a boilerplate gadget tick code that you can use to define the callback method.

Boilerplate gadget tick code:

```
//Gadget defined field: _lastProcessedTimerTick
_lastProcessedTimerTick = null,

//Gadget defined field: _maxTimerCallbackThreshold
_maxTimerCallbackThreshold = 500,

//Gadget defined field: _forceTickProcessingEvery (10 seconds)
_forceTickProcessingEvery = 10000,

/**
  * Processes a timer tick - updating the UI.
  * @param start is the time that the tick was received
  * @returns {boolean} true
  */
_processTick = function (start) {

   //Developer's add UI update logic here
   //...
   //...

   _lastProcessedTimerTick = start;

  return true;
},

/**
  * Timer tick callback handler.
  * @param data
  */
_timerTickHandler  = function (timerTickEvent) {
   var start, end, diff, discardThreshold, processed;

  start = (new Date()).getTime();
  processed = false;

  //Prevent starvation of timer logic
  if (_lastProcessedTimerTick === null) {
     processed = _processTick(start);
  } else {
     if ((_lastProcessedTimerTick + _forceTickProcessingEvery) <= start) {
       //Force processing at least every _forceTickProcessingEvery milliseconds
       processed = _processTick(start);
     }

  end = (new Date()).getTime();
  diff = end - start;
  if (diff > _maxTimerCallbackThreshold) {
    _clientLogs.log("GadgetXYZ took too long to process timer tick (_maxTimerCallbackThreshold
 exceeded).");
```

```
   }
},
```

If you choose not to use the boilerplate gadget tick code, you should ensure the following:

- Callback calculates entry and exit time.
- Callback for timer tick is quick (log when callback takes to long - only when exceeding threshold).
- Callback provides discard capability (as outlined in the boilerplate gadget tick code) to prevent events from piling up.
- Callback adds a _lastProcessedTimerTick and uses it to force an update to occur at regular intervals (such as every 10 seconds). The intent is to prevent starvation in a heavily-loaded system that cannot respond quickly enough, such that all events are being discarded.

Note Because the timer callback triggers every 1 second and the JavaScript engine is single-threaded, it is important to process as quickly as possible. Using the boilerplate code makes gadget development issues more obvious and easier to debug.

Handling Special Characters in CSS

When using CSS in a gadget, the Finesse Desktop Gadget Container restricts the following special characters:

@ ^ $ * :: ~

If the CSS contains any of the special characters listed above, copy the following JavaScript code into your gadget's *.js file:

```
/**
 * Injects css or js files into DOM dynamically.
 * This is to bypass gadget container's restriction for special chars in CSS 3 files.
 * E.g. @Keyframes
 */
injectResource : function (url){
     var node = null;
     // url null? do nothing
     if(!url) {
        return;
     }
     // creates script node for .js files
     else if(url.lastIndexOf('.js')=== url.length-3){
         node = document.createElement("script");
         node.async = false;
         node.setAttribute('src', url);
     }
     // creates link node for css files
     else if(url.lastIndexOf('.css')== url.length-4){
         node = document.createElement("link");
         node.setAttribute('href', url);
         node.setAttribute('rel', 'stylesheet');
     }
     // inserts the node into dom
     if(node) {
       document.getElementsByTagName('head')[0].appendChild(node);
```

```
        }
 }
```

In your gadget's *.xml file, call the injectResource function that you have copied above. The parameter to the injectResource function is the path to your css file:

```
<script type="text/javascript">
       <your gadget namespace>.injectResource('<path to CSS file>/<CSS filename>.css');
</script>
```

Subscription Management on Finesse Desktop

Because the Finesse desktop provides a managed XMPP connection to the Cisco Finesse Notification Service, the ability to subscribe or unsubscribe to a particular notification feed is also provided as an interface using the SubscribeNodeReq and UnsubscribeNodeReq requests described in Finesse Requests.

Gadget Height Management

The height of the gadget is managed in several ways. As the gadget is essentially an iFrame HTML element, the height of the gadget refers to the height of the iFrame. This height can be set and modified by the following options.

- Finesse desktop layout XML (fixed height)
- Gadget API (dynamic height)

Setting Gadget Height—Desktop Layout XML

The gadget height is provided in pixels as a query parameter to the gadget URL in the Finesse desktop layout XML. The query parameter used in the XML is `gadgetHeight`. This is recommended if the height of the content inside the gadget does not change frequently.

In the following example, the initial height of the gadget, when it gets rendered on the client, is set to 150 pixels.

Example

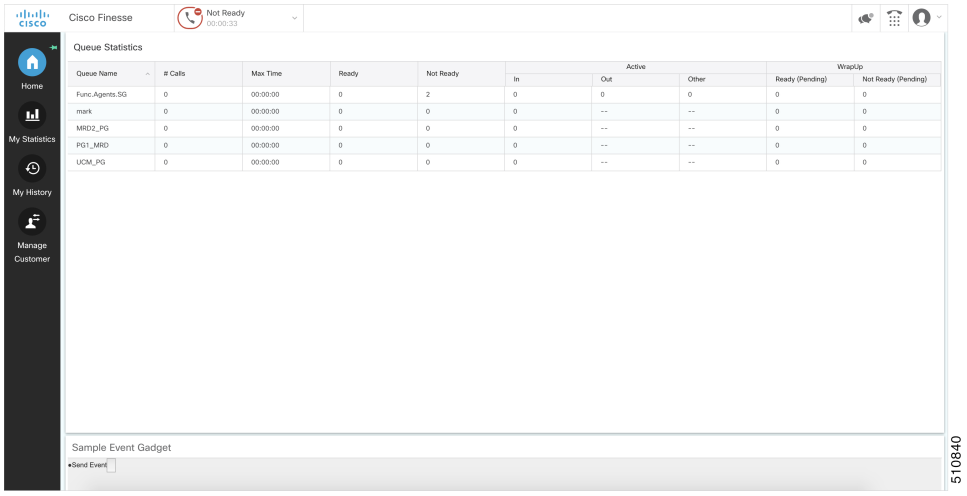
```
<gadget>https://my-server.com/gadgets/SampleGadget/SampleGadget.jsp?gadgetHeight=150</gadget>
```

In the above example, the gadget content cannot define the height of the gadget. You would want the gadget to take the height of the content dynamically. But since the gadget is an iFrame, it has limited capabilities to adjust height automatically. Setting the gadget height through Finesse desktop layout XML gives you a gadget with fixed height with no resizing capabilities.

Setting Gadget Height—Using Gadget API

You might have a gadget whose content is dynamic, such as a grid that is being populated with data dynamically. The number of rows can increase or decrease in real-time. If you have a gadget with highly dynamic content, then fixed height may result in a lot of extra white space or there is not enough space to show the whole content inside the gadget. These issues can be resolved with the `gadgets.window.adjustHeight(opt_height)` API.

Figure 12: Extra White Space

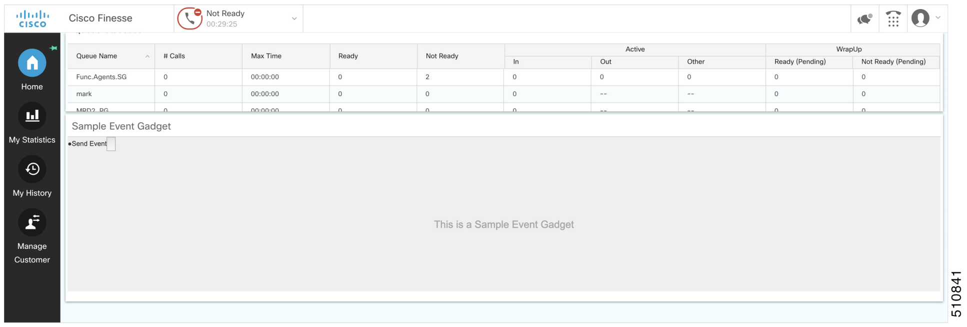


Queue Statistics

Queue Name	# Calls	Max Time	Ready	Not Ready	Active			WrapUp	
					In	Out	Other	Ready (Pending)	Not Ready (Pending)
Func.Agents.SG	0	00:00:00	0	2	0	0	0	0	0
mark	0	00:00:00	0	0	0	--	--	0	0
MRD2_PG	0	00:00:00	0	0	0	--	--	0	0
PG1_MRD	0	00:00:00	0	0	0	--	--	0	0
UCM_PG	0	00:00:00	0	0	0	--	--	0	0

Sample Event Gadget

Figure 13: Space Constraint



Queue Name	# Calls	Max Time	Ready	Not Ready	Active			WrapUp	
					In	Out	Other	Ready (Pending)	Not Ready (Pending)
Func.Agents.SG	0	00:00:00	0	2	0	0	0	0	0
mark	0	00:00:00	0	0	0	--	--	0	0
MRD2_PG	0	00:00:00	0	0	0	--	--	0	0

Sample Event Gadget

This is a Sample Event Gadget

You can set the height of the gadget dynamically from inside the gadget itself by passing the height in pixels to the `gadgets.window.adjustHeight(opt_height)` API.

For example, consider a gadget which shows the details of a customer currently on the call. This gadget is supposed to have two views to display the information.

Minimal view—Displays minimum information which is default option. This displays only 10 fields out of 50 which are the most critical information about the customer such as, customer id, first name, last name, and email address.

Enlarged view—Displays maximum information which displays all the 50 fields related to the customer information.

A simple button can be used to toggle these views. The minimal view takes less space. Hence a gadget with lesser height would suffice. The enlarged view takes more space to fit in the content in a user-friendly manner (no or minimal scrollbars).

Example of Toggle View

```
var view = 'minimal'; // view is minimal by default
function toggleView() {
    if (view === 'minimal') {
        view = 'enlarged';
        // add more fields to the DOM here and do something else which is also required.
       // Once you have the DOM updated with the content use the adjustHeight API to adjust
 the height of the iFrame in px according your need
```

```
        gadgets.window.adjustHeight(500);
    } else {
        // here we are going back to minimal view
        view = 'minimal';
        // remove the extra fields from here and do something else which is also required.
       // Once you have the DOM updated with the content use the adjustHeight API to adjust
 the height of the iFrame in px according your need
        gadgets.window.adjustHeight(100);
    }
}
```

In the above example, the dynamic nature of the view is limited to two fixed sizes, one is the minimal view which can always fit in an iFrame of height 100 pixels and the other is the enlarged view with more details which can fit in an iFrame of height 500 pixels.

Calling the gadgets.window.adjustHeight() without Parameter

Consider a gadget with a grid which displays the real-time data of the signed-in users in a team to a Supervisor. This grid would dynamically update whenever a user of that team signs-in or signs-out. The size of the grid can vary from one row to n number of rows. To calculate the height of the gadget with respect to the changing number of rows in the grid you must call the `gadgets.window.adjustHeight(opt_height)` API without any parameter.

Whenever `gadgets.window.adjustHeight(opt_height)` API is called without the height parameter, it calculates the height of the content inside the iFrame and applies that height to the iFrame. It is recommended that the gadget calls this API in the gadget code, which can manipulate the DOM to change the size of the content inside it.

Example of Adjust Height without Parameter

```
team.getUsers({
    onLoad: function(users) {
        // load the grid first time
        for (user in users.getCollection()) {
            if (user.getState() === 'LOGGED_IN') {
                // render each row of logged in users
            }
        }
        // calling the adjustHeight API will automatically calculate the height of the
content and apply it to the iFrame
        gadgets.window.adjustHeight();
    },
    onCollectionAdd: function(user) {
        // add the newly added user to the grid
        if (user.getState() === 'LOGGED_IN') {
            // add this row to the grid
        }
       // adjusts the height each time a row is added, so that the content is fully visible
 in the iFrame
        gadgets.window.adjustHeight();
    }
});
```

CHAPTER 9

Third-Party Gadgets

Enable or Reset 3rdpartygadget Account

Use the following CLI command to enable (or reset) the password for the 3rdpartygadget account:

utils reset_3rdpartygadget_password

You are prompted to enter a password. After you enter a password, you are prompted to confirm the password.

You must set the password before you can upload gadgets using SFTP.

Note You must enable or reset the password for the 3rdpartygadget account on install. The password must be between 5 and 32 characters long and must not contain spaces or double quotes (").

CSS Requirements

By default, Finesse rewrites the linked CSS in your gadget, which in some cases is not desirable as it results in a loss of functionality if the CSS you are loading refers to other asynchronous elements. As a result, for all third-party gadgets, you can bypass the content rewriting for CSS by including the following in your gadget XML:

1. Add the optional feature "content-rewrite" to disable the CSS rewrite:

```
<Optional feature="content-rewrite">
        <Param name="expires">86400</Param>
        <Param name="include-url">.*</Param>
        <Param name="exclude-url">.css</Param>
</Optional>
```

2. Include UserPref for "externalServerHost":

   ```
   <UserPref name="externalServerHost"/>
   ```

3. To reference the CSS file, perform one of the following:

   - If the gadget is hosted on the Finesse server, reference the CSS file using externalServerHost:

     ```
     <link rel="stylesheet"
     href="__UP_externalServerHost__/3rdpartygadget/files/<yourgadgetname>/<path to CSS
     file>/<CSS filename>.css"
     type="text/css"/>
     ```

     where you must update `<yourgadgetname>` to the filename of your gadget under the 3rdpartygadget `/files` folder and update the remaining path variables to the location of the CSS file for your gadget.

   - If the gadget is hosted on a server external to Finesse, reference the CSS file using the URL:

     ```
     <link rel="stylesheet"
     href="[https:]//<hostname>/<path to CSS file>/<CSS filename>.css"
     type="text/css"/>
     ```

     where you must update the URL variables to the location of the CSS file on your external server, and where specifying the protocol (https) is optional. (If you omit the protocol, Finesse uses the default protocol of the page.)

Note Finesse Desktop Gadget Container restrains special characters while loading a CSS3 file. See Handling Special Characters in CSS, on page 444

Upload Third-Party Gadgets

After you set the password for the 3rdpartygadget account, you can use SFTP to upload third-party gadgets to the Finesse server, as illustrated in the following example. Note that third-party gadget files must be .xml files. It does not support .jsp files.

Note Finesse allows you to upload third-party gadgets to your own web server, however, you must ensure that the Finesse server has access to your web server.

```
my_workstation:gadgets user$ sftp 3rdpartygadget@<finesse>
3rdpartygadget@<finesse>'s password:
Connected to <finesse>.
sftp> cd /files
sftp> put HelloWorld.xml
Uploading HelloWorld.xml to /files/HelloWorld.xml
```

```
HelloWorld.xml
sftp> exit
```

After you upload a gadget, it is available under the following URL:

https://<finesse>/3rdpartygadget/files/

To access the gadget uploaded in the previous example, use the following URL:

https://<finesse>/3rdpartygadget/files/HelloWorld.xml

When you add a gadget to the desktop layout, that gadget can be referenced using a relative path. For more information on adding third party gadgets to the Finesse desktop layout, see the section *Manage Desktop Layout* in the *Cisco Finesse Administration Guide*.

To include the gadget that was uploaded in the previous example in the desktop layout, add the following XML (highlighted) to the layout:

```
<finesseLayout xmlns="http://www.cisco.com/vtg/finesse">
      <layout>
        <role>Agent</role>
        <page>
          <gadget>/desktop/gadgets/CallControl.jsp</gadget>
          <gadget>/3rdpartygadget/files/HelloWorld.xml</gadget>
        </page>
        ...
      </layout>
      <layout>
        <role>Supervisor</role>
        <page>
          <gadget>/desktop/gadgets/CallControl.jsp</gadget>
          <gadget>/3rdpartygadget/files/HelloWorld.xml</gadget>
        </page>
        ...
      </layout>
    </finesseLayout>
```

Note You cannot delete, rename or change permissions of a folder while using SFTP in 3rd party gadget accounts for Unified CCX deployments. To perform these actions, SELinux has to be in permissive mode. This can be accomplished by running the following CLI command:

utils os secure permissive

Note Because of browser caching and caching in the Finesse web server, you may need to clear the browser cache or restart the Cisco Finesse Tomcat service before gadget changes take effect. If you make a change to a gadget and the change is not reflected on the Finesse desktop, clear your browser cache.

If you do not see the changes after you clear the browser cache, use the following CLI command to restart the Cisco Finesse Tomcat service:

admin:utils service restart Cisco Finesse Tomcat

Third-Party Gadget Limitations

Third-party gadgets must be .xml files. You cannot use .jsp files.

Permissions

If a newly uploaded third-party gadget does not render via the desktop layout or when you launch it directly in a browser, the gadget files may not have the correct permissions. If gadget files do not have read permissions for everyone else (for example, the file permission is 770), Cisco Finesse Tomcat cannot read them. The minimum file permission should be 644.

If a gadget file does not have the correct permissions, when you launch it directly in the browser, you receive a 404 “Resource not available” error. When you try to launch the gadget via the desktop layout, you receive an error message that states the requested resource is not available.

To change file permissions on the Finesse server, use SFTP (CLI or client program) as shown in the following example:

```
$ sftp 3rdpartygadget@172.27.184.59
3rdpartygadget@172.27.184.59's password:
Connected to 172.27.184.59.
sftp> cd files
sftp> ls -l
---------- 1 751 751 0 Dec 6 19:40 MyGadget.xml
sftp> chmod 644 MyGadget.xml
Changing mode on /files/MyGadget.xml
sftp> ls -l
-rw-r--r-- 1 751 751 0 Dec 6 19:40 MyGadget.xml
sftp>
```

Replication

You must set the password for the 3rdpartygadget account on both the primary and secondary Finesse servers.

Gadgets must be manually uploaded to both the primary and secondary Finesse servers.

Migration

When you perform an upgrade, third-party gadgets are migrated to the new version.

The 3rdpartygadget account password is not migrated across upgrades. After an upgrade, you must reset the password for the 3rdpartygadget account before you can make changes to third-party gadgets. You must reset the password on both the primary and secondary Finesse servers.

Backup and Restore

Third-party gadgets are preserved when you perform a DRS backup and restore.

Restrictions

Any attempt to GET JavaServer Pages (jsp) using the URL https://<finesse>/3rdpartygadget/files is blocked. You will receive a 403 (Access Denied) error code when attempting to retrieve a jsp.

CORS Support for Finesse REST APIs

Cross-Origin Resource Sharing (CORS) is a verification mechanism that uses additional HTTP headers to let a user gain permission to access selected resources from a server on a different origin (domain) than the site currently in use. By default, CORS support is disabled for Cisco Finesse and Cisco Finesse Notification Service. The CORS support can be enabled by the Administrator for specific origins listed in the allowed origin list using CLIs. For more information see, *Cisco Finesse Admin guide 12.0(1)* located at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-user-guide-list.html. CORS requests that are originating from the allowed origin list will be honored as per CORS RFC.

Maintenance Mode

Third-party gadget developers can register with client services to receive notifications (*FINESSE_MAINTENANCE_MODE_EVENT*) when maintenance mode is scheduled. These notifications are triggered 15 seconds before the desktop reload. The DesktopCache API can be used to save and restore the gadget state after reloading to the alternate node of the Cisco Finesse server.

If a third-party gadget is logged in to a voice/non-voice Media Routing Domain (MRD), the gadget must send a `NOT_READY` message with reason code: 50045 to the Finesse server after a successful migration during maintenance mode. This message does not change the agent state.

CHAPTER 10

Cisco Finesse JavaScript APIs

Client Services

Class finesse.clientservices.ClientServices

Allows clients to make the Cisco Finesse API requests and use Cisco Finesse events by using the JavaScript functions that are provided by this module. This service layer establishes a notification connection that is shared between all the gadgets and the desktop for its eventing needs. It uses events for client subscriptions and constructs API requests.

Methods

getNotificationConnectionState()

Retrieves the current state of the BOSH/WebSocket connection.

Example

```
finesse.clientservices.ClientServices.getNotificationConnectionState();
```

Returns

`{String}` The current state of the BOSH/WebSocket connection from the following options:.

- Connected—When the connection is established between client and openfire.
- Recovered —When the connection is re-established after a failure.

- Failing —When the notification service or the Cisco Finesse server is down.

getRestHost()

Retrieves the destination host where the REST requests are proxied through Shindig. This method can be used before making REST requests to retrieve the hostname.

Example

```
finesse.clientservices.ClientServices.getRestHost();
```

Returns

`{String}` The hostname of Cisco Finesse.

init(config)

Initiates the ClientServices module with the specified configuration parameters. For more information, see Gadget Configuration, on page 492.

Example

```
finesse.clientservices.ClientServices.init(finesse.gadget.Config)
```

Throws

`{Error}` If the valid parameter is missing during initialization.

registerOnConnectHandler(handler)

Adds a handler to be invoked when the following conditions are met:

- Cisco Finesse goes IN_SERVICE wherein all the operations of Cisco Finesse is performed or accepted.
- BOSH/WebSocket connection is established and the client application communicates with the Cisco Finesse Notification Service through BOSH/WebSocket to receive notifications. The loss of this connection means that the server is UNAVAILABLE or that the client cannot reach the server.
- Cisco Finesse user presence becomes available. The presence indicates whether Finesse has an active connection to the Cisco Finesse Notification Service (Unified CCE) or the Cisco Unified CCX Notification Service (Unified CCX). An UNAVAILABLE presence for the Cisco Finesse XMPP user means that the connection is lost.

For more information.

If these conditions are met when this function is called, the handler is invoked immediately.

Example

```
_cs = finesse.clientservices.ClientServices;
_cs.registerOnConnectHandler(_connectionConnectHandler);

_connectionConnectHandler = function () {
 // Perform the logic
}
```

Parameters

Name	Type	Description	Required
handler	Function	The function that is invoked when the conditions are met. Registers only one handler at a time. Handlers registered earlier are overwritten.	Yes

registerOnDisconnectHandler(handler)

Adds a handler or callback to be invoked when any of the following occurs:

- Cisco Finesse is no longer IN_SERVICE
- BOSH/WebSocket connection is lost
- Cisco Finesse user presence becomes UNAVAILABLE

If any of these conditions are met at the time this function is called, the callback is invoked immediately.

Example

```
_cs = finesse.clientservices.ClientServices;
_cs.registerOnDisconnectHandler(_connectionDisconnectHandler);

_connectionDisconnectHandler = function () {
 // Perform the logic
}
```

Parameters

Name	Type	Description	Required
handler	Function	The function that is invoked when the conditions are met. Registers only one handler at a time. Handlers registered earlier are overwritten.	Yes

Container Services

Class finesse.containerservices.ContainerServices

Provides container level services for gadget developers. Gadgets can utilize the container dialogs and event handling to add or remove a service.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
containerServices.addHandler(
    finesse.containerservices.ContainerServices.Topics.ACTIVE_TAB,
    function() {
        clientLogs.log("Gadget is now visible"); // log to Finesse logger
        // automatically adjust the height of the gadget to show the html
        gadgets.window.adjustHeight();
    }
);
containerServices.makeActiveTabReq();
```

Methods

activateMyTab()

Activates the tab in the container in which the gadget is present.

Example

```
finesse.containerservices.ContainerServices.activateMyTab();
```

activateTab(tabId)

Activates a particular tab in the container.

Example

```
finesse.containerservices.ContainerServices.activateTab(tabId);
```

Parameters

Name	Type	Description	Required
tabId	String	The Id (not the label text) of the tab is activated. If the Id is invalid, no action occurs.	Yes

addHandler(topic, callback)

Adds a handler for the specific Container Services topic. For more information on topics, see Container Services Topics, on page 466.

Example

```
finesse.containerservices.ContainerServices.addHandler(finesse.containerservices.ContainerServices.Topics.
TIMER_TICK_EVENT,callback);
```

Parameters

Name	Type	Description	Required
topic	String	Hub topic that the gadget wants to listen to. For more information on topics, see Container Services Topics, on page 466.	Yes
callback	Function	An asynchronous callback function that is invoked when the hub topic is notified.	Yes

collapseMyGadget()

Collapses the gadget by hiding its contents for gadgets that are collapsible. In the collapsed state, only the gadget header is displayed.

Example

```
finesse.containerservices.ContainerServices.collapseMyGadget();
```

To make the gadget collapsible, add `<Optional feature="collapsible" />` in to the ModulePrefs inside the gadget XML.

Example: Enable Collapse Feature

```
<ModulePrefs title="Sample Gadget" description="Sample Gadget">
    <Optional feature="collapsible" />
</ModulePrefs>
```

Note Ensure that the gadget height is set using `gadgets.window.adjustHeight` before making a call to `finesse.containerservices.ContainerServices.collapseMyGadget`.

enableTitleBar()

Displays the title bar for a page-level gadget which is not visible by default. This is not applicable for a tab-level gadget and can be used within a gadget.

Example

```
containerServices = finesse.containerservices.ContainerServices.init();
containerServices.enableTitleBar();
```

expandMyGadget()

Expands the gadget which is collapsed to display its contents.

Example

```
finesse.containerservices.ContainerServices.expandMyGadget();
```

Note Ensure that the gadget height is set using `gadgets.window.adjustHeight` before making a call to `finesse.containerservices.ContainerServices.expandMyGadget`.

getMyGadgetId()

Retrieves the Id of the gadget.

Example

```
finesse.containerservices.ContainerServices.getMyGadgetId();
```

Returns

`{Number}` Id of the gadget

getMyGadgetView()

Returns the current view details of the gadget. To identify if the gadget is in canvas (maximized) or default (restored) state, you must use this at the gadget inital load time. In all the other scenarios, gadget can use `finesse.containerservices.ContainerServices.Topics.GADGET_VIEW_CHANGED_EVENT` to get to the gadget view change events.

Example

```
containerServices = finesse.containerservices.ContainerServices.init();
var viewConfig = containerServices.getMyGadgetView();
var newgadgetHeight = viewConfig.maxAvailableHeight;
```

Returns

`{Object}` The gadget details that include the following:

- gadgetId—The Finesse gadget ID
- tabId—The tab ID of the container or the gadget
- maxAvailableHeight—Maximum available height that can be used by the gadget iframe
- view—'canvas' or 'default', where canvas is maximized and default is restored state of a gadget

getMyTabId()

Retrieves the `tabId` of the container or gadget.

Example

```
finesse.containerservices.ContainerServices.getMyTabId();
```

Returns

`{String}` The `tabId` of the container or gadget.

hideCertificateBanner(id)

Hides the Certificate Banner. The banner is hidden when all the gadgets that invoked `showCertificateBanner` have made a corresponding invocation to `hideCertificateBanner`, or when the user closes the banner manually.

Example

```
// For Gadgets
containerServices = finesse.containerservices.ContainerServices.init();
containerServices.hideCertificateBanner();

// For non gadget Client
containerServices.hideCertificateBanner(id);
```

Parameters

Name	Type	Description	Required
id	String	System generated unique ID to identify a banner. This `id` parameter is used by the desktop when the banner has to be invoked by a non-gadget client. Gadgets do not send this parameter.	Yes, when invoked by a non-gadget client

hideDialog()

Hides the user interface modal dialog.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
containerServices.hideDialog();
```

hideMyGadget()

Makes the current gadget inaccessible by hiding it from the Multi-Tab gadget header.

Note This operation has no effect and does not cause any errors when the gadget is not hosted within a Multi-Tab gadget.

Example

```
finesse.containerservices.ContainerServices.hideMyGadget();
```

hideMyGadgetNotification()

Removes the current gadget's notifications from the Multi-Tab gadget header.

Note This operation has no effect and does not cause any errors when the gadget is not hosted within a Multi-Tab gadget.

Example

```
finesse.containerservices.ContainerServices.hideMyGadgetNotification();
```

init()

Initiates the ContainerServices module.

Note The init method must be called before using the ContainerServices object for invoking any other functionality.

Example

```
finesssе.containerServices.ContainerServices.init();
```

Returns

`{finesse.containerServices.ContainerServices}` The initiated finesse.containerServices.ContainerServices reference.

isTabbedGadget()

Checks if the gadget is configured inside a multi-tab gadget.

Example

```
containerServices = finesse.containerservices.ContainerServices.init();
containerServices.isTabbedGadget();
```

Returns

`{Boolean}` True if the gadget is hosted inside a multi-tab gadget.

makeActiveTabReq()

Requests to activate the tab in which the gadget is present.

Example

```
finesse.containerservices.ContainerServices.makeActiveTabReq();
```

publish(topic, data)

Publishes data to the specified topic on the OpenAjax hub. Since gadgets reside in different iFrames, message publication is the only way to communicate with each other. This method gives a mechanism for the gadgets to have their data published through custom topics, which in turn can be used by other gadgets by subscribing to the same topic.

For example, OnClick is an element in one gadget which triggers an action in another gadget that is achieved by using this method.

Example

```
finesse.containerservices.ContainerServices.publish('CUSTOMTOPIC', data);
finesse.containerservices.ContainerServices.addHandler("CUSTOMTOPIC", function(data) {
    // Perform the logic
});
```

Parameters

Name	Type	Description	Required
topic	String	Hub topic that the gadget wants to listen to. For more information on topics, see Container Services Topics, on page 466.	Yes
data	Object	The data to be published for the specified topic on the OpenAjax hub.	Yes

reloadMyGadget()

Reloads the current gadget. This method is useful when the gadget encounters an error.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
containerServices.reloadMyGadget();
```

reloadMyGadgetFromUrl(url)

Updates the URL for this gadget and reloads the gadget. This method allows the gadget to be reloaded from a different URL which can be useful for third-party gadgets implementing a failover mechanism.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
containerServices.reloadMyGadgetFromUrl(url);
```

Parameters

Name	Type	Description	Required
url	String	The URL that the gadget should reload from.	Yes

removeHandler(topic, callback)

Removes previously added handler for the specified Container Services topic.

Example

```
finesse.containerservices.ContainerServices.removeHandler(finesse.containerservices.ContainerServices.Topics.
TIMER_TICK_EVENT,callback);
```

Parameters

Name	Type	Description	Required
topic	String	Hub topic that the gadget wants to listen to. For more information on topics, see Container Services Topics, on page 466.	Yes
callback	Function	An asynchronous callback function to be removed for the specified Container Services topic.	No

setMyGadgetTitle(title)

Sets the title of the current gadget.

Example

```
finesse.containerservices.ContainerServices.setMyGadgetTitle('Recent History');
```

Parameters

Name	Type	Description	Required
title	String	The title of the current gadget.	Yes

showCertificateBanner(callback)

Displays the Certificate Banner with the message `Gadget certificates are yet to be accepted.`

Example

```
// For Gadgets
containerServices = finesse.containerservices.ContainerServices.init();
containerServices.showCertificateBanner(function(){
    // Do something when the banner hides
});

// For non gadget Client , id is required to hide the certificate banner
// which is returned when showCertificateBanner is called
var id = containerServices.showCertificateBanner(function(){
    // Do something when the banner hides
});
```

Parameters

Name	Type	Description	Required
callback	Function	Callback is invoked when user closes the banner manually.	Yes

Returns

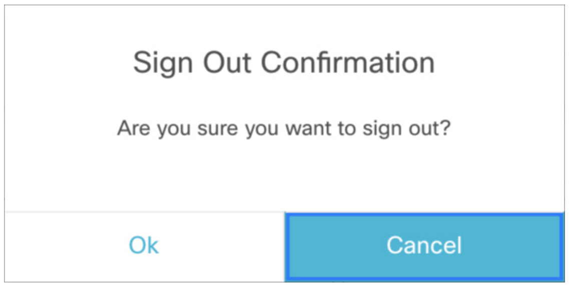
`{String}` id—System generated unique ID to identify a banner. This `id` parameter is used by the desktop when the banner has to be invoked by a non-gadget client. Gadgets do not send this parameter.

showDialog(options)

Shows the user interface modal dialog with the specified parameters. The parameters are:

- Title of the modal dialog
- Message inside the modal dialog
- Label for the button to close the modal dialog
- If the modal dialog should block other dialogs
- If the modal dialog is draggable
- If the modal dialog is fixed-size

Figure 14: Sample UI Modal Dialog


Sign Out Confirmation

Are you sure you want to sign out?

Note Custom JavaScript-based modal dialogs and alerts negatively affects the functionality of the Finesse desktop and is not recommended to be used.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
containerServices.showDialog({
    title: 'Error Occurred',
    message: 'Something went wrong',
    close: function() {
    }
});
```

Parameters

Name	Type	Description	Required
title	String	Title of the modal dialog.	Yes

Name	Type	Description	Required
options	Object	Options for the modal dialog. • close—Callback function that gets invoked when the close button of the modal dialog is clicked. • message—Message to be displayed in the modal dialog. • isBlocking—Indicates whether the modal dialog blocks other dialogs from being shown.	Yes

Returns

`{Object}` The modal dialog object of the modal dialog DOM (Document Object Model) element.

showMyGadget()

If hidden, makes the tab corresponding to this gadget visible in the Multi-Tab gadget.

Note This operation has no effect and does not cause any errors, when the gadget is not hosted within a Multi-Tab gadget.

Example

```
finesse.containerservices.ContainerServices.showMygadget();
```

showMyGadgetNotification()

When hosted in a Multi-Tab gadget, a notification will appear on the tab corresponding to this gadget.

Note This operation has no effect and does not cause any errors, when the gadget is not hosted within a Multi-Tab gadget.

Example

```
finesse.containerservices.ContainerServices.showMyGadgetNotification();
```

```
finesse.containerservices.ContainerServices.showMyGadgetNotification(messageDetails);
```

messageDetails is a method that has the following parameters:

```
messageFrom, message, isDismissable, timeout, icon, type, pristine
```

tabVisible()

Retrieves the visibility of the current gadget only after the initialization of the gadget.

Example

```
finesse.containerservices.ContainerServices.tabVisible();
```

Returns

`{Boolean}` The visibility of the gadget.

Container Services Topics

Class finesse.containerservices.ContainerServices.Topics

Set of topics used for subscribing events from ContainerServices. The method to subscribe to topics is `finesse.containerservices.ContainerServices.containerServices.addHandler();`. For more information, see addHandler(topic, callback), on page 458.

Table 10: Field Details

Topic Name	Description
ACTIVE_CALL_STATUS_EVENT	Listens to an active call event. Callback is invoked when an agent voice state changes from Ready or Not Ready to any other non-callable state or vice versa. There are two types of responses: • Active call—ActiveCallStatusEvent {status: true, type: "info"} • End or inactive call—ActiveCallStatusEvent {status: false, type: "info"}
ACTIVE_TAB	Listens to changes to the active tab. Callback is invoked when the tab containing the gadget becomes active. The method to use when the gadget is in the active tab finesse.containerservices.ContainerServices.makeActiveTabReq()
FINESSE_MAINTENANCE_MODE_EVENT	Listens to notifications related to Finesse maintenance mode changes. Callback is invoked when the desktop migration is scheduled or about to happen. Callback contains the status of the maintenance. For more information see the section Maintenance Mode, on page 453 There are two types of statuses: • SCHEDULED: Notified when the migration is scheduled by the Cisco Finesse server. • MIGRATING: Notified that the agent is migrating after 15 seconds.

Topic Name	Description
GADGET_VIEW_CHANGED_EVENT	Listens to changed events of the gadget view. Callback is invoked when a gadget view changes. There are two types of views: • Default (set by the developer) • Canvas (full-screen view) The callback passes finesse.containerservices.GadgetViewChangedEvent. For more information, see *Finesse Desktop Gadget Development* section in *Cisco Finesse Web Services Developer Guide* at https://developer.cisco.com/docs/finesse/#!rest-api-dev-guide.
TIMER_TICK_EVENT	Listens to the TimerTick event. Callback is invoked when this event is run. Cisco Finesse publishes TimerTickEvent to OpenAjax hub every 1000 milliseconds. The callback passes finesse.containerservices.TimerTickEvent. For more information, see *Finesse Container Timer* section in *Cisco Finesse Web Services Developer Guide* at https://developer.cisco.com/docs/finesse/#!rest-api-dev-guide.
WORKFLOW_ACTION_EVENT	Listens to workflow action traffic events. When the trigger and the conditions defined for a workflow are completed, then a workflow action event is published, which is used to run the workflow action. The callback passes finesse.containerservices.WorkflowActionEvent. For more information, see *Workflow Action Event* section in *Cisco Finesse Web Services Developer Guide* at https://developer.cisco.com/docs/finesse/#!rest-api-dev-guide.

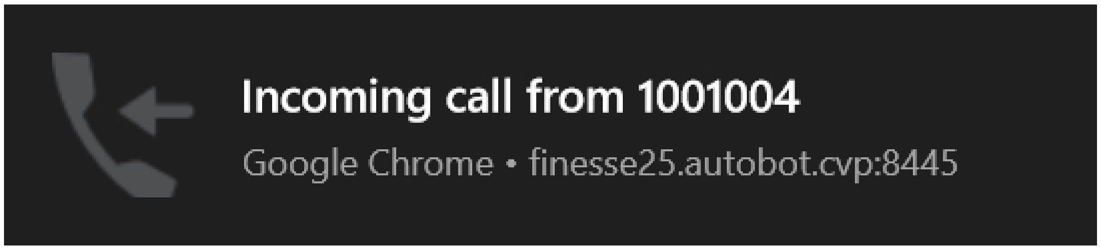
Finesse Toaster

Class finesse.containerservices.FinesseToaster

FinesseToaster is a utility class that displays Cisco FinesseToaster notifications. FinesseToaster is a built-in browser notification that appears at the bottom of the screen, and is typically used to notify the user of important events when the agent desktop browser tab is not active. FinesseToaster uses the HTML5 Notification API to display the notification. For more details on HTML5 Notification API and browser compatibility, see https://developer.mozilla.org/en-US/docs/Web/API/notification#Browser_compatibility.

Note Internet Explorer does not support the toaster functionality.

Figure 15: Sample Finesse Toaster Notification



Incoming call from 1001004

Google Chrome • finesse25.autobot.cvp:8445

Methods

init(config, logger)

Initiates the Cisco Finesse Toaster module for the gadget to be able to display notifications.

Example

```
finesse.containerservices.FinesseToaster.init("config,logger");
```

Parameters

Name	Type	Description	Required
config	Object	The configuration data which is either the finesse.container.Config or finesse.gadget.Config.	Yes
logger	Object	The finesse.cslogger.ClientLogger object for the client logging messages. For example, you can use finesse.cslogger.ClientLogger as a parameter.	No

Returns

`{finesse.containerservices.FinesseToaster}` The initiated finesse.containerServices.FinesseToaster reference.

showToaster(title, options)

Displays Cisco FinesseToaster notification to the user.

Example

```
finesse.containerservices.FinesseToaster.showToaster(
    'Incoming Alert', {
        body: 'There is new message'
    }
);
```

Parameters

Name	Type	Description	Required
title	String	The title of the Cisco FinesseToaster notification.	Yes
options	Object	Options for the Cisco FinesseToaster notification. • body—The text in the Cisco FinesseToaster notification • icon—The URL of the image for the icon in the Cisco FinesseToaster notification Cisco FinesseToaster notification default icons. The constant lists are: • TOASTER_DEFAULT_ICONS.INCOMING_CALL_ICON • TOASTER_DEFAULT_ICONS.INCOMING_CHAT_ICON • TOASTER_DEFAULT_ICONS.INCOMING_TEAM_MESSAGE • autoClose—Duration in milliseconds, the Cisco FinesseToaster notification remains opened. The default value is 8000 milliseconds. **Note** The autoClose parameter is applicable only for Chrome and Firefox browsers in Windows OS. In macOS, Chrome and Firefox browsers automatically close the toaster notification. In macOS, the autoClose value is ignored and the browser automatically closes the toaster notification. • showWhenVisible—Determines how the Cisco Finesse Toaster notification is displayed based on the visibility of the Cisco Finesse desktop. • true—Shows the Cisco Finesse Toaster notification irrespective of whether the Cisco Finesse desktop is active or not. • false (default)—Shows the Cisco Finesse Toaster notification only when the Cisco Finesse desktop is inactive.	Yes

Popover Service

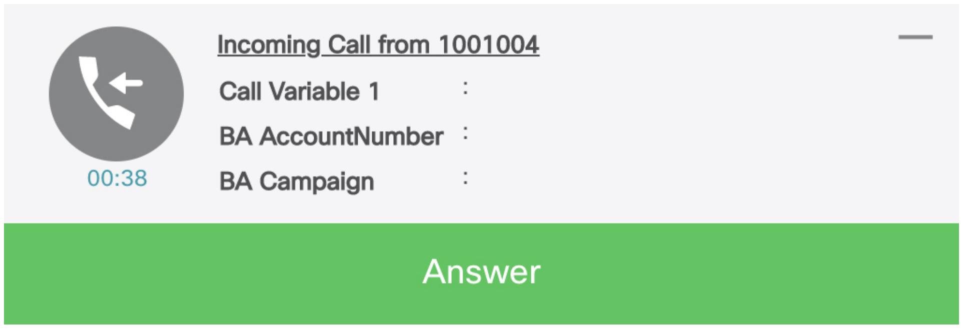
Class finesse.containerservices.PopoverService

Cisco Finesse voice component and gadgets hosting digital services uses the finesse.containerservices.PopoverService to display a popover for incoming calls and chat events.

Note

The popover is different from Cisco Finesse toaster notification. Toaster is a built-in browser notification, and it appears only if the agent desktop's browser tab is in the background.

Figure 16: Sample Popover to Answer a Call



Object Definitions

The finesse.containerservices.PopoverService class uses specific data objects as inputs. The properties and their values of the object are JSON Schema compliant. The format is defined below.

actionData

The actionData object defines the set of actions that can be taken and the buttons to be displayed popover.

```
finesse.containerservices.PopoverSchema.getActionDataSchema()
```

Sample actionData Object

```
actionData = {
    "dismissible": false,
    "keepMaximised": false,
    "clientIdentifier": 'popup1', // A string to uniquely identify a specific popover
    "requiredActionText": "Please answer the call from your phone",
    "buttons": // Optional. Max 2
        [{
            "id": "No",
            "label": "Decline",
            "type": "Affirm",
            "hoverText": "NOOO",
            "confirmButtons": [ // confirmButtons is an optional property in actionData
                {
                    "id": "Yes",
                    "label": "Reject - Return to campaign",
                    "hoverText": "YESSSS"
                },
                {
                    "id": "No",
                    "label": "Close - Remove from campaign",
                    "hoverText": ""
                }
            ]
        }]
};
```

The payload details are explained in the table below.

Key	Type	Description
dismissible	Boolean	Determines whether the popover must be manually dismissed. • True—Popover to be dismissed automatically after the specified time is lapsed. • False—User to take action on the displayed popover based the buttons defined. For example, Answer or Decline.
keepMaximised	Boolean	Determines whether the popover must be maximized or minimized. • True—Shows the popover minimized when there is more than one popover. • False—Shows the popover maximized when there is more than one popover.
clientIdentifier	String	Unique identifier across all popovers. Used in the callback for popover events.
requiredActionText (optional)	String	The text in the popover that describes the user action.
buttons (optional)	Array	Options for the action buttons (maximum two) in the popover.
-->id	String	Unique identifier across all popovers. Used in the callback for popover events.
-->label	String	The text of the action button.
-->type	Enum	The color of the button. • Affirm—Green button refers to affirm • Decline—Light gray button refers to decline.
-->hoverText (optional)	String	The tooltip when you hover the mouse pointer over the 'requiredActionText' (popover) when the text is truncated.
-->confirmButtons (optional)	Object	The confirmation message with the buttons in response to the user action.
—->id	String	Unique identifier of the confirmation button.
—->label	String	The label on the button.
—->hoverText	String	The tooltip message on the confirmation button.

The following method can be used to get the schema for the actionData object.

```
finesse.containerservices.PopoverSchema.getActionDataSchema().
```

bannerData

The bannerData object helps to configure the data displayed on the popover.

```
finesse.containerservices.PopoverSchema.getBannerDataSchema()
```

Sample bannerData Object

```
bannerData = {
    "icon": { // Mandatory
        "type": "collab-icon",
        "value": "chat"
    },
    "content": [ // Optional. first 6 name/value pairs is shown in popover
        {
            "name": "Customer Name",
            "value": "Michael Littlefoot"
        },
        {
            "name": "Phone Number",
            "value": "+1-408-567-789"
        },
        {
            "name": "Account Number",
            "value": "23874567923"
        },
        {
            "name": "Issue", // For the below one, tool tip is displayed
            "value": "a very long text."
        }
    ],
    "headerContent": {
        "maxHeaderTitle": "Popover maximised title",
        "minHeaderTitle": "Popover minimized title"
    }
};
```

The payload details are explained in the table below.

Key	Type	Description
icon	Object	The icon displayed in the popover.
-->type	Enum	The type of icon in the popover. For more information, see Cisco Common Desktop Stock Icon Names with Image, on page 487.
–>value	String	The display name of the icon.
content (optional)	Array	The list of six names or value pairs to be displayed in the popover.
-->name	String	The display name of an individual name or value pair to be displayed on the popover.
-->value	String	The corresponding value of the name for the individual name or value pair to be displayed on the popover.
headerContent	Object	The title of the popover when it is maximized or minimized.
-->maxHeaderTitle	String	The title of the popover when it is maximized.
-->minHeaderTitle	String	The title of the popover when it is minimized.

The following method can be used to get the schema for the bannerData object.

```
finesse.containerservices.PopoverSchema.getBannerDataSchema()
```

timerData

The timerData object helps configure the timer that displayed on the popover, which indicates the time left for the popover to be dismissed.

```
finesse.containerservices.PopoverSchema.getTimerDataSchema()
```

Sample timerData Object

```
timerData = {
    "displayTimeoutInSecs": 60,
    "display": true, // false means no displayable UI for timer
    "counterType": 'COUNT_UP'
}
```

The payload details are explained in the table below.

Key	Type	Description
displayTimeoutInSecs (mandatory)	Integer	The popover timeout in seconds. The minimum is 3 seconds and the maximum is 3600 seconds. –1 refers to no upper limit.
display	Boolean	Determines whether the timer must be displayed on the popover. • True—Shows the time left for the popover to be dismissed. • False—Shows the popover without the time left for dismissal.
counterType	Enum	Determines the direction in which time in the popover should be updated. • COUNT_UP—Shows the time elapsed for taking action on the popover, and the timer begins counting up. • COUNT_DOWN— Shows the time left for taking action on the popover, and the timer begins counting down.

The following method can be used to get the schema for the timerData object.

```
finesse.containerservices.PopoverSchema.getTimerDataSchema()
```

Methods

dismissPopover(popoverId)

Dismisses the popover with the given popover Id.

Parameters

Name	Type	Description	Required
popoverId	String	Unique identifier of the popover to be dismissed. This Id is returned from the showPopover call.	Yes

Throws

`{Error}` If the service is not initialized.

generatePayload(isExistingPopover, popoverId, bannerData, timerData, actionData)

Generates a single payload for use by the popover service.

Parameters

Name	Type	Description	Required
isExistingPopover	Boolean	Determines whether the popover is shown in the Finesse desktop. • True—Shows the popover in the Finesse desktop. • False—Does not show the popover in the Finesse desktop.	Yes
popoverId	String	Unique identifier of the popover to be generated. This Id is returned from the showPopover call.	Yes
bannerData	Object	The data displayed on the popover. For example, Customer Information such as name and phone number. For more information on the payload and description, see bannerData, on page 472.	Yes
timerData	Object	The time left for the popover to be dismissed. The duration is displayed in seconds. For example, 00:38. For more information on the payload and description, see timerData, on page 473.	Yes
actionData	Object	Describes the set of actions to be taken on the displayed popover. For example, Answer and Decline. For more information on the payload and description, see actionData, on page 470.	Yes

Throws

`{Error}` If the popoverData is not as per defined format.

init(ContainerService)

Initiates the PopoverService which is used by gadgets.

Parameters

Name	Type	Description	Required
ContainerService	Function	Provides container level services for gadget developers. Gadgets can utilize the container dialogs and event handling to add or remove a service.	Yes

Returns

`{finesse.containerservices.PopoverService}` The initiated finesse.containerServices.PopoverService reference.

showPopover(bannerData, timerData, actionData, actionHandler)

Shows a popover with the specified data. The user interaction or timeout of the popover is notified to the gadget through the registered actionHandler.

Note When there is a new popover, the older popover is minimized except the popover related to voice calls.

Parameters

Name	Type	Description	Required
bannerData	Object	The data displayed on the popover. For example, Customer Information such as name and phone number. For more information on the payload and description, see bannerData, on page 472.	Yes
timerData	Object	The time left for the popover to be dismissed. The duration is displayed in seconds. For example, 00:38. For more information on the payload and description, see timerData, on page 473.	Yes
actionData	Object	Describes the set of actions to be taken on the displayed popover. For example, Answer and Decline. For more information on the payload and description, see actionData, on page 470.	Yes
actionHandler	Function	Handler function that gets invoked for the events that are associated with the user interactions.	Yes
-->popoverId	String	Unique identifier of the popover to be generated. This Id is returned from the showPopover call.	Yes
-->source	String	Unique identifier of the source which generates the event. For example, 'btn_[id]_click', 'dismissed', or 'timeout'.	Yes

Throws

`{Error}` If the popoverData is not as per defined format.

Returns

`{String}` The popover Id and can be used for subsequent interaction with the service.

updatePopover(popoverId, bannerData, timerData, actionData, actionHandler)

Updates an active popover's displayed content.

Parameters

Name	Type	Description	Required
popoverId	String	Unique identifier of the popover to be generated. This Id is returned from the showPopover call.	Yes

Name	Type	Description	Required
bannerData	Object	The data displayed on the popover. For example, Customer Information such as name and phone number. For more information on the payload and description, see bannerData, on page 472.	Yes
timerData	Object	The time left for the popover to be dismissed. The duration is displayed in seconds. For example, 00:38. For more information on the payload and description, see timerData, on page 473.	Yes
actionData	Object	Describes the set of actions to be taken on the displayed popover. For example, Answer and Decline. For more information on the payload and description, see actionData, on page 470.	Yes
actionHandler	Function	Handler function that gets invoked for the events that are associated with the user interactions.	Yes

Throws

`{Error}` If the popoverData is not as per defined format.

Events

Gadget View Changed Event

Class finesse.containerservices.GadgetViewChangedEvent

Contains information about the changed events of the gadget view. There are two types of views supported for a gadget, that is default and canvas. The default view is the default size of the gadget. Canvas view is a maximized view of the gadget.

You can set a button to toggle between default view and canvas view. Add `<Content type="html" view="default,canvas">` in the gadget XML. This adds the toggle button in the top right corner of the gadget. For more information, see Gadget Height Management, on page 445.

The method to subscribe to the changed gadget event is `finesse.containerservices.ContainerServices.containerServices.addHandler()` with a topic of `finesse.containerservices.ContainerServices.Topics.GADGET_VIEW_CHANGED_EVENT`.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
finesse.containerservices.ContainerServices.addHandler(finesse.containerservices.
ContainerServices.Topics.GADGET_VIEW_CHANGED_EVENT, function(gadgetViewChangedEvent) {
            var gadgetId = finesse.containerservices.ContainerServices.getMyGadgetId(),
                tabId = finesse.containerservices.ContainerServices.getMyTabId();
            if (gadgetViewChangedEvent.getGadgetId() === gadgetId &&
gadgetViewChangedEvent.getTabId() === tabId) {
                if (gadgetViewChangedEvent.getView() === 'default') {
                    gadgets.window.adjustHeight(defaultHeight);
                    $('#content').html('DEFAULT VIEW');
                    view = 'default';
                } else if (gadgetViewChangedEvent.getView() === 'canvas') {

gadgets.window.adjustHeight(gadgetViewChangedEvent.getMaxAvailableHeight());
                    $('#content').html('CANVAS VIEW');
                    view = 'canvas';
                }
            }
```

Methods

getGadgetId()

Retrieves the gadget Id.

Returns

`{String}` Unique Identifier for the gadget changing view.

getMaxAvailableHeight()

Retrieves the maximum available height of the gadget.

Returns

`{String}` The maximum available height for the gadget's view.

getTabId()

Retrieves the tab Id.

Returns

`{String}` Unique Identifier for the tab where the gadget changing view resides.

getView()

Retrieves the gadget view.

Returns

`{String}` The view type of the gadget.

Timer Tick Event

Class finesse.containerservices.TimerTickEvent

Contains information about the events of the timer-tick. The method to subscribe to the changed gadget event is `finesse.containerservices.ContainerServices.addHandler()` with a topic of `finesse.containerservices.ContainerServices.Topics.TIMER_TICK_EVENT()`.

When the gadget is attaching a handler for the time ticker topic, it is called periodically with tick frequency mentioned. By default, the value is one second. For more information, see Finesse Container Timer, on page 442.

Example

```
finesse.containerservices.ContainerServices.addHandler(finesse.containerservices.
ContainerServices.Topics.TIMER_TICK_EVENT,updateTimer);
```

Method

getDateQueued()

Retrieves the TimerTickEvent *dateQueued* field.

Returns

`{Date}` The date object when the TimerTickEvent is queued.

Workflow Action Event

Class finesse.containerservices.WorkflowActionEvent

Contains information about the events of the workflow action. Gadgets subscribe to the finesse.containerservices.workflowActionEvent to receive workflow action events to run as a result of workflow evaluations. For more information, see Workflow Action Event, on page 441.

The method to subscribe to the workflow event is `finesse.containerservices.ContainerServices.containerServices.addHandler()` with a topic of `finesse.containerservices.ContainerServices.Topics.WORKFLOW_ACTION_EVENT`. Gadgets must listen to events with the handleBy value of "OTHER", which is configured through `cfadmin`. Selecting "OTHER" in cfadmin that implies the action is run by other third-party gadgets and not Finesse desktop. The handleBy value is fetched by using the function `getHandledBy()`.

For more information, see the *Manage Workflows* chapter in *Cisco Finesse Administration Guide* at https://www.cisco.com/c/en/us/support/customer-collaboration/finesse/products-maintenance-guides-list.html.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
containerServices.addHandler("finesse.containerservices.workflowActionEvent",
function(workflowActionEvent) {
    var type, handledby
    params = workflowActionEvent.getParams();
    actionVariabless = workflowActionEvent.getActionVariables();
    handledby = workflowActionEvent.getHandledBy();
    type = workflowActionEvent.getType();
    //
    if (handledby === "OTHER" && type === "BROWSER_POP") {
        // Have the logic
    }
});
```

Methods

getActionVariables()

Retrieves the WorkflowActionEvent action variables map.

Returns

`{Object}` The object for the action variables map, where key is the action variable name, and value is the Object such as name, type, node, testValue, and actualValue.

getHandledBy()

Retrieves the WorkflowActionEvent handledBy value. Gadgets search for events with a handleBy value of OTHER.

Returns

`{String}` The handledBy value of the WorkflowAction which is finesse.containerservices.WorkflowActionEvent.HandledBy.

getName()

Retrieves the WorkflowActionEvent name.

Returns

`{String}` The name of the WorkflowAction.

getParams()

Retrieves the WorkflowActionEvent parameters map.

Returns

`{Object}` The object for the parameters map, where key is the param name, and value is the Object such as name, value, and expandedValue ().

The type of the WorkflowAction are BROWSER_POP and HTTP_REQUEST.

BROWSER_POP

- windowName—Name of the window as seen on the browser tab header.
- path—URL to open.

HTTP_REQUEST

- method—PUT or POST.
- location—FINESSE or OTHER.
- contentType—If applicable, then MIME type of request body. For example, text or plain.
- path—Request URL.
- body—Request content for POST requests.

getType()

Retrieves the WorkflowActionEvent type.

Returns

`{String}` The type of the WorkflowAction (BROWSER_POP or HTTP_REQUEST).

Workflow Action Event.HandledBy

Class finesse.containerservices.WorkflowActionEvent.HandledBy

Contains information about the set of possible HandledBy values used for WorkflowActionEvent from ContainerServices. This is provided from the finesse.containerservices.WorkflowActionEvent.getHandledBy method.

Field Details

Name	Description
FINESSE	The Cisco Finesse handles the WorkflowActionEvent. The third-party does the additional processing with the action. Cisco Finesse handles this WorkflowAction.
OTHER	The third-party handles the WorkflowActionEvent. Cisco Finesse's Workflow Engine program ignores the action and expects the Gadget Developers to take action.

Task Activity Notification

Note Cisco Finesse TaskActivity API enables gadgets on the Finesse desktop to listen to and provide information about the user's multi-channel task-related activity in Unified CCE. Task activity status corresponding to multiple media that agents are signed in to can be published or subscribed by using this API.

Unlike traditional desktop APIs, the information provided by this API is only produced and consumed by gadgets on the Finesse desktop. Therefore using this API requires supporting gadgets that can consume or produce this information to be deployed.

Class finesse.containerservices.TaskActivityNotification

Provides a framework to notify and receive updates about the digital task activity status. The notifications inform the desktop and other subscribers about which non-voice media dialog is currently active or inactive.

Note This is supported from Cisco Finesse, Release 12.5(1) ES3 onwards.

Example

```
var taskActivityNotification =
finesse.containerservices.TaskActivityNotification.init(containerServices);
```

Methods

init()

Initializes the TaskActivityNotification object.

Example

```
var taskActivityNotification =
finesse.containerservices.TaskActivityNotification.init(containerServices);
```

Returns

`{finesse.containerservices.TaskActivityNotification}` The initialized finesse.containerservices.TaskActivityNotification reference.

notifyTaskSelection(from, message)

Notifies the subscribing gadgets about a task becoming active or inactive through the OpenAjax hub. The *from* parameter uniquely identifies the publishing gadget, and the *message* parameter contains the payload that is being published, with the details of the task.

Example

```
var taskActivityNotification =
finesse.containerservices.TaskActivityNotification.init(containerServices);

var from = "Gadget_id";

var message = {
    'timestamp': 1589780515222, // optional
    'taskId': 'task_id'
    'active': true,
    'mediaType': 'Chat',
    'contextInfo': {
        // optional
    }
}

taskActivityNotification.notifyTaskSelection(from, message);
```

Parameters

Name	Type	Description	Required
from	String	An identifier for the gadget that publishes notifications.	Yes
message	String	The task activity selection message to be published to the OpenAjax hub.	Yes
-->timestamp	Number	The Unix Epoch timestamp at which the message was generated. If the value is not present, Finesse automatically populates it with the time at which the API request was made.	Optional
-->taskId	String	Unique identifier for the task. **Note** The taskID used must correspond to the ID of an existing Finesse MediaDialog. This requirement exists since gadgets on the Finesse desktop does not know about tasks outside of those provided by the digital channel APIs.	Yes
-->active	Boolean	Determines the task activity status. • True—If the task is active. • False—If the task is inactive.	Yes
-->mediaType	String	The type of media under which the dialog is classified. For example, Chat and Email.	Yes
-->contextInfo	Object	Any additional information in JSON format.	Optional

registerForTaskNotifications(callback)

Registers the gadget for receiving task activity notifications issued by other gadgets, through the callback provided as a parameter to the registration.

Example

```
var taskActivityNotification =
finesse.containerservices.TaskActivityNotification.init(containerServices);

// register to receive task notifications
taskActivityNotification.registerForTaskNotifications(_processTaskNotifications);
```

Parameters

Name	Type	Description	Required
callback	Function	Callback function that is invoked when a task activity notification message is received.	Yes

requestCurrentTasks()

Request to receive the last published task activity notifications from all gadgets that are actively publishing task notifications. This API is intended to be used when gadgets interested in task notifications initializes itself. Therefore, the current task activity status, which might have been missed while the gadget was initializing or loading, can be received. This API must be invoked only after registering for notifications.

Example

```
var taskActivityNotification =
finesse.containerservices.TaskActivityNotification.init(containerServices);

// register to receive task notifications
taskActivityNotification.registerForTaskNotifications(_processTaskNotifications);

// request for a task notification
taskActivityNotification.requestCurrentTasks();
```

Throws

`{Error}` If the callback function is not registered using `TaskActivityNotification.registerForTaskNotifications`.

ClientLogger

Class finesse.cslogger.ClientLogger

Allows gadgets to send the client log messages over the hub by calling the log method of the clientLogger. This enables the container to collect the logs of the third-party gadget and make it available on the server.

Methods

init(hub, gadgetId, config)

Initiates the client logger object for the client logging messages.

Example

```
var _clientLogger = finesse.cslogger.ClientLogger;
_clientLogger.init(gadgets.Hub, "MyGadgetId", config);
```

Parameters

Name	Type	Description	Required
hub	Object	The Shindig hub topic that the gadgets wants to listen to.	Yes
gadgetId	String	Unique identifier of the gadget.	Yes
config	Object	The configuration data which is used to get the hostname for the third-party gadget.	Yes

log(message, error)

Publishes a log message over the hub.

Example

```
_clientLogger.log("This is a important message for MyGadget");
```

Parameters

Name	Type	Description	Required
message	String	The log message displayed in the hub.	Yes
error	Object	The message that is associated with the error.	Optional

Digital Channel

Digital channels are distinct and separate from the voice state control of Cisco Finesse. For example, Email and Chat. Digital channels are intended for gadgets to represent their custom channels and can be programmatically used to control the digital channel state.

The Finesse Digital Channel State Control (FNC), a programmable desktop component, is available from Cisco Finesse Release 12.0(1) onwards. This API provides the schema that is used in `finesse.digital.ChannelService` for various digital channel operations.

Figure 17: Component Interaction

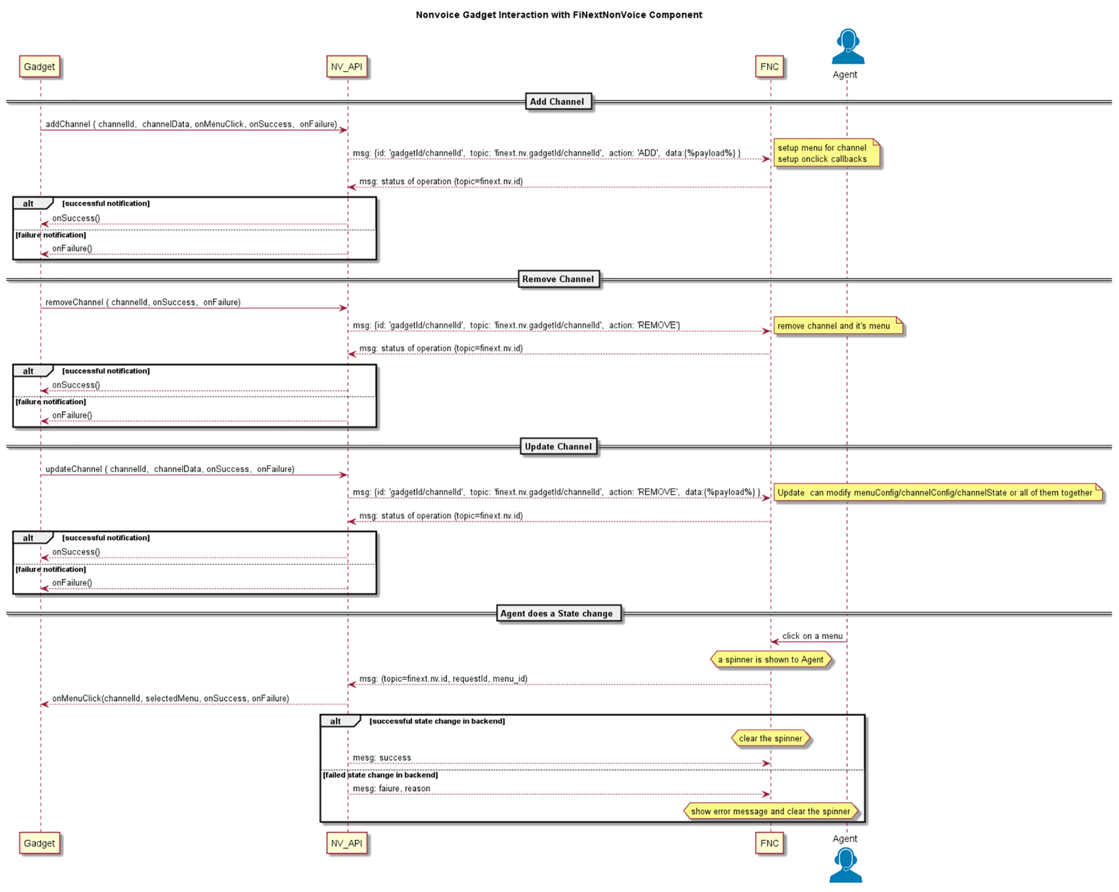
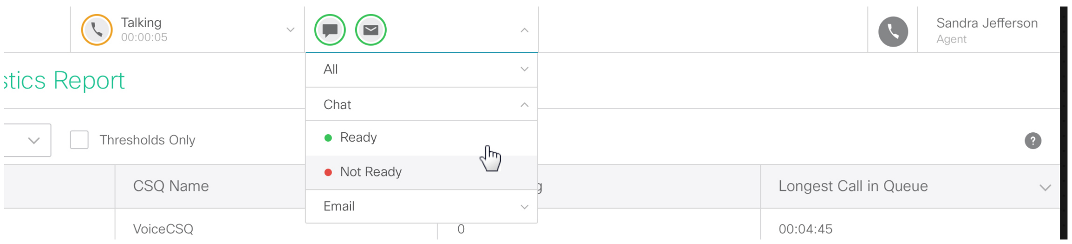

Nonvoice Gadget Interaction with FiNextNonVoice Component
Gadget
NV_API
FNC
Agent
Add Channel
addChannel (channelId, channelData, onMenuClick, onSuccess, onFailure)
msg: {id: 'gadgetId/channelId', topic: 'finext.nv.gadgetId/channelId', action: 'ADD', data:{%payload%} }
setup menu for channel
setup onclick callbacks
msg: status of operation (topic=finext.nv.id)
alt [successful notification]
onSuccess()
[failure notification]
onFailure()
Remove Channel
removeChannel (channelId, onSuccess, onFailure)
msg: {id: 'gadgetId/channelId', topic: 'finext.nv.gadgetId/channelId', action: 'REMOVE'}
remove channel and it's menu
msg: status of operation (topic=finext.nv.id)
alt [successful notification]
onSuccess()
[failure notification]
onFailure()
Update Channel
updateChannel (channelId, channelData, onSuccess, onFailure)
msg: {id: 'gadgetId/channelId', topic: 'finext.nv.gadgetId/channelId', action: 'REMOVE', data:{%payload%} }
Update can modify menuConfig/channelConfig/channelState or all of them together
msg: status of operation (topic=finext.nv.id)
alt [successful notification]
onSuccess()
[failure notification]
onFailure()
Agent does a State change
click on a menu
a spinner is shown to Agent
msg: (topic=finext.nv.id, requestId, menu_id)
onMenuClick(channelId, selectedMenu, onSuccess, onFailure)
alt [successful state change in backend]
clear the spinner
mesg: success
[failed state change in backend]
mesg: faiure, reason
show error message and clear the spinner
Gadget
NV_API
FNC
Agent
510668

Figure 18: Digital Channel Options



Object Definitions

The finesse.digital.ChannelService class uses specific data objects as inputs. The properties and their values of the object are JSON Schema compliant. The format is defined below.

channelConfig

The channelConfig object helps to configure the digital channel details.

```
finesse.containerservices.ChannelSchema.getChannelConfigSchema()
```

Sample channelConfig Object

```
{
    "actionTimeoutInSec": 5,
    "icons": [{
            "type": "collab-icon",
            "value": "Chat"
        },
        {
            "type": "url",
            "value": "../../thirdparty/gadget3/channel-icon.png"
        }
    ]
}
```

The payload details are explained in the table below.

Key	Type	Description
actionTimeoutInSec	Integer	The duration for which the FNC waits after sending the menu selection request to the gadget. The duration is mentioned in seconds, and the upper limit is 30 seconds. During this period, no other operation can be performed on the FNC.
icons	Array	The icons displayed in the header to represent a digital channel.
-->type	Enum	The type of the icon in the header. For more information, see Cisco Common Desktop Stock Icon Names with Image, on page 487.
-->value	String	The display name of the icon.

The following method can be used to get the schema for the channelConfig object.

```
finesse.containerservices.ChannelSchema.getChannelConfigSchema()
```

channelState

The channelState object defines the state of the digital channel.

```
finesse.containerservices.ChannelSchema.getChannelStateSchema()
```

Sample channelState Object

```
{
    "label": "Chat & Email",
    "currentState": "ready",
    "iconColor": "available",
    "enable": true,
    "logoutDisabled": true,
    "logoutDisabledText": "Please go unavailable on chat before logout",
    "iconBadge": "none"
    "hoverText": "Tooltip text"
}
```

Key	Type	Description
label	String	The label of the digital channel.
currentState	String	The text that describes the current state of the digital channel.

Key	Type	Description
iconColor	Enum	The color of the icon based on the current state of the digital channel. • Green color refers to available. • Red color refers to unavailable. • Orange color refers to busy.
enable	Boolean	Determines whether the digital channel is shown in the Finesse desktop. • True—Shows the digital channel menu in the Finesse desktop. • False—Does not show the digital channel menu in the Finesse desktop.
logoutDisabled	Boolean	Determines whether the logout menu is shown in the user identity component. • True—Disables the logout menu in the user identity component. • False—Enables the logout menu in the user identity component. For example, during Agent Ready or Busy state, disable the logout menu and enable it again when the state changes to Not Ready.
logoutDisabledText	String	The text to the user if the logout menu is disabled.
iconBadge	String	The type of badge based in the digital channel. • *info* refers to the information badge. • *error* refers to an error badge. • *warning* refers to the warning badge. • *none* refers to no badge.
hoverText	String	The tooltip when you hover the mouse pointer.

The following method can be used to get the schema for the channelState object.

```
finesse.containerservices.ChannelSchema.getChannelStateSchema()
```

menuConfig

The menuConfig object helps to configure the menu details.

```
finesse.containerservices.ChannelSchema.getMenuConfigSchema()
```

Sample menuConfig Object

```
{
    "label": "Chat",
    "menuItems": [{
            "id": "ready-menu-item",
            "label": "Ready",
            "iconColor": "available"
```

```
        },
        {
            "id": "not-ready-menu-item",
            "label": "Not Ready",
            "iconColor": "unavailable"
        }
    ]
}
```

Key	Type	Description
label	String	The label of the digital channel.
menuItems	Array	The list of the menu items for the digital channel.
-->id	String	Unique identifier of the digital channel menu. When there is a user action on the digital channel menu, this Id is returned through the parameter `finesse.digital.ChannelService.selectedMenuItemId`.
-->label	String	The text of the menu item.
-->iconColor	Enum	The color of the icon based on the current state of the digital channel. • Green color refers to available. • Red color refers to unavailable. • Orange color refers to busy.

The following method can be used to get the schema for the menuConfig object.

```
finesse.containerservices.ChannelSchema.getMenuConfigSchema()
```

Cisco Common Desktop Stock Icon Names with Image

The digital channel configuration schema considers the Cisco Common Desktop icon (CD-icon) name as its value. The icons are composed of different elements. Sign in to Cisco Finesse and paste the following JavaScript code in the editor of your browser developer console to see the list of CD-UI icon names and their visual design. This script cleans the Cisco Finesse web page, displays the icon name, and renders it in an HTML table. Refresh the browser to reflect the changes.

Note You can also define this value in a gadget.

Example

```
var showIcons = function() {
    $('body').html('');

    $('body').append("<table border='1' background-color:#a0c0a0;'>" +
        "<thead style='display: none;'><th>Icon Name</th>" +
        "<th>Icon</th></thead><tbody  " +
        "style='display: block;  overflow-y: auto; height: 600px'>" +
        "</tbody></table>");

    var icons = window.top.cd.core.cdIcon;
```

```
        var addIcon = function(name, iconJson) {

            var width = (iconJson.width) ? iconJson.width : 1000;
            var height = (iconJson.height) ? iconJson.height : 1000;

            var iconBuilt = "<tr><td>" + name +
                "</td><td><svg width='" + width +
                "' height='" + height +
                "' style='height: 30px; width: 30px;'  viewBox='" +
                iconJson.viewBox + "'>" +
                iconJson.value + "</svg></td></tr>";


            try {
                $('tbody').append(iconBuilt);
            } catch (e) {
                console.error("Error when adding " + name, e);
            }
        }

        for (var icon in icons) {
            if (icons[icon].viewBox) addIcon(icon, icons[icon])
        }
    }

    showIcons()
```

Channel Service

Class finesse.digital.ChannelService

Provides methods that are leveraged by the gadgets serving digital channels to register, update, or modify digital channel-specific display information and corresponding menu action behavior in Agent State Control Menu (referred to as the FNC Menu component).

These APIs are available to the gadget through the finesse.min.js import. For more information on how to write a sample gadget, see https://github.com/CiscoDevNet/finesse-sample-code/tree/master/LearningSampleGadget.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
channelService = finesse.digital.ChannelService.init(containerServices);
channelService.addChannel(channelId, channelData, onMenuClick, onSuccess, onError);
```

Field Details

Name	Description
ICON_BADGE_TYPE	The type of badge in the digital channel.
ICON_TYPE	The type of icon in the digital channel. For more information, see Cisco Common Desktop Stock Icon Names with Image, on page 487.
STATE_STATUS	The color of the icon based on the current state of the digital channel.
STATUS	The operation status of the digital channel.

Methods

addChannel(channelId, channelData, onMenuClick, onSuccess, onError)

Add a digital channel to the FNC menu component. The API requires the complete digital channel state in the form of a JSON payload. Developers must pre-validate the JSON against its corresponding schema by testing it through finesse.utilities.JsonValidator.validateJson. The result of the add operation is returned through the given success or error callback.

Example

```
finesse.digital.ChannelService.addChannel(channelId, channelData, onMenuClick, onSuccess,
onError);
```

Parameters

Name	Type	Description	Required
channelId	String	Unique identifier to register the digital channel with FNC. Used in the callback for FNC.	Yes
channelData	Object	The data of the key-value pair added to the digital channel as JSON payload. The following are the channelData keys: • menuconfig • channelConfig • channelState For more information on the channelData keys, see Digital Channel, on page 483.	Yes
onMenuClick	Function	Callback function that is invoked when the menu button of the digital channel is clicked.	Yes
onSuccess	Function	Callback function that is invoked upon a successful add operation.	Yes
onError	Function	Callback function that is invoked upon an unsuccessful add operation.	Yes

Success payload has the following format:

```
 {
    "channelId": "[ID of the Digital channel]",
    "status": "success"
}
```

Error payload has the following format:

```
{
    "channelId": "[ID of the Digital channel]",
    "status": "failure",
    "error": {
        "errorCode": "[Channel supplied error code that will be logged in Finesse client
logs]",
        "errorDesc": "An error occurred while processing request"
    }
}
```

Throws

`{Error}` If the digital channelData passed on is not as per the schema.

init(ContainerServices)

Initiates the ChannelService module.

Example

```
finesse.digital.ChannelService.init(finesse.containerservices.ContainerServices);
```

Parameters

Name	Type	Description	Required
ContainerServices	Function	Provides container level services for gadget developers. Gadgets can utilize the container dialogs and event handling to add or remove a service.	Yes

Returns

`{finesse.digital.ChannelService}` The initiated finesse.digital.ChannelService reference.

removeChannel(channelId, onSuccess, onError)

Removes the previously added digital channel representation from the FNC menu component.

Example

```
finesse.digital.ChannelService.removeChannel(channelId, onSuccess, onError);
```

Parameters

Name	Type	Description	Required
channelId	String	Unique identifier of the digital channel to be removed. This Id is returned from the FNC.	Yes
onSuccess	Function	Callback function that is invoked upon a successful remove operation.	Yes
onError	Function	Callback function that is invoked upon an unsuccessful remove operation.	Yes

updateChannel(channelId, channelData, onSuccess, onError)

Updates the digital channel in the FNC menu component. None of the data that is passed within the data payload channelData is mandatory. This API provides an easy way to update the complete channel configuration in one go or partially if necessary. The result of the update operation is intimated through the given success and error callbacks.

Example

```
finesse.digital.ChannelService.updateChannel(channelId, channelData, onSuccess, onError);
```

Parameters

Name	Type	Description	Required
channelId	String	Unique identifier of the digital channel to be removed. This Id is returned from the FNC.	Yes
channelData	Object	The data of the key-value pair updated to the digital channel as JSON payload. For more information on the object description, see addChannel(channelId, channelData, onMenuClick, onSuccess, onError), on page 489	Yes
onSuccess	Function	Callback function that is invoked upon a successful update operation.	Yes
onError	Function	Callback function that is invoked upon an unsuccessful update operation.	Yes

updateChannelMenu(channelId, menuItems, onSuccess, onError)

Updates the menu displayed for the digital channel.

Example

```
finesse.digital.ChannelService.updateChannelMenu(channelId, menuItems, onSuccess, onError);
```

Parameters

Name	Type	Description	Required
channelId	String	Unique identifier of the digital channel to be removed. This Id is returned from the FNC.	Yes
menuItems	Array	The list of menu items for the digital channel. For more information, see menuConfig, on page 486.	Yes
onSuccess	Function	Callback function that is invoked upon a successful update operation.	Yes
onError	Function	Callback function that is invoked upon an unsuccessful update operation.	Yes

updateChannelState(channelId, channelState, onSuccess, onError)

Updates the digital channel's current state.

Example

```
finesse.digital.ChannelService.updateChannelMenu(channelId, channelState, onSuccess, onError);
```

Parameters

Name	Type	Description	Required
channelId	String	Unique identifier of the digital channel to be updated. This Id is returned from the FNC.	Yes
channelState	Object	The current state of the digital channel. For more information, see channelState, on page 485.	Yes

Name	Type	Description	Required
onSuccess	Function	Callback function that is invoked upon a successful update operation.	Yes
onError	Function	Callback function that is invoked upon an unsuccessful update operation.	Yes

Gadget Configuration

Class finesse.gadget.Config

Provides configuration data from the container page for the gadgets Config object in the Finesse desktop. When Finesse desktop load successfully, the container loads configuration details from the server. While creating gadgets, these configurations are passed to the gadgets as `finesse.gadget.Config` object. For more information on gadgets.

Field Details

Field	Description
authorization	The Base64 encoded "id:password" that can be used for non-SSO authentication by the gadget to make Finesse REST API requests.
authToken	The token used for authentication in SSO deployments.
clientDriftInMillis	The time difference between the client and the server in milliseconds.
compatibilityMode	The configuration of the client compatibility mode. This field is used to display a message to the user or handle compatibility mode use cases in the gadget. The compatibility mode in Internet Explorer is a feature that helps you to view webpages that were designed for the previous versions of the browser. Enabling this feature affects the newer sites that were designed for modern browsers.
country	The country code of the client derived from the locale.
extension	The extension with which the user signs in.
host	The Finesse server IP address or host.
hostPort	The connections and listening port of the Finesse server host.
id	The unique identifier of the signed-in agent, which is used to uniquely identify an agent using the Cisco Finesse REST API URI and in the desktop notifications.
language	The language code of the client derived from the locale.
locale	The locale of the client.
localhostFQDN	The fully qualified domain name of the local host.
localhostPort	The connections and listening port of the local host.

Field	Description
mobileAgentDialNumber	The phone number that the system calls to connect with the mobile agent.
mobileAgentMode	The work mode of the mobile agent that is found in `finesse.restservices.User.WorkMode`.
peripheralId	Unique identifier of the CTI server peripheral that Finesse is connected to.
pubsubDomain	The pub sub domain of the XMPP service where the pub subservice is running.
restHost	The IP address or host of the Finesse API.
scheme	The type of HTTP protocol (http or https).
skillTargetId	Unique identifier for the skill target assigned to the user in the Unified CCE Database. It is supported from Cisco Finesse, Release 12.5(1) ES2 onwards. **Note** This is only supported for Unified CCE deployments.
systemAuthMode	The system authorization mode of the Finesse deployment.
teamId	Unique identifier of the team that the user belongs to.
teamName	The name of the team that the user belongs to.
toasterNotificationTimeout	The duration in seconds, the Cisco Finesse toaster notification remains opened.
xmppDomain	The domain of the XMPP service.

Interfaces

Request Handlers

Class finesse.interfaces.RequestHandlers(handlers)

Defines the REST object callback handlers that are passed as arguments while creating the REST object. Retrieves the methods when the object is created.

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered. The following are the request handlers (see below for details): • success(rsp) • error(rsp)	Optional

Name	Type	Description	Required
success(rsp)	Function	Callback function that is invoked upon a successful request. The initialized object is then passed to the callback function as a parameter.	Optional
error(rsp)	Function	Callback function that is invoked upon an unsuccessful request. The initialized object is then passed to the callback function as a parameter.	Optional

status Number The HTTP status code of the succeeded request. Optional content String The raw string response of the succeeded request. Optional object Object The parsed object response of the succeeded request. Optional error Object The error details from the failed request. Optional errorType String The type of error. Optional errorMessage String The message that is associated with the error. Optional

REST Services

JavaScript Representation of Finesse REST API

Finesse JavaScript library uses JavaScript objects that represent the underlying REST API objects[1] such as a User, Dialog, Phonebook and so on. When a Finesse JavaScript class is initialized, a corresponding REST API call is made, and the response is populated into a JavaScript object. In addition to having JavaScript object representations of Finesse REST API objects, the Finesse JavaScript library also supports the subscription to the Finesse Notification Service. When a Finesse notification is sent for a particular JavaScript object, the corresponding handler of the object is triggered with the updated JavaScript object as the parameter.

1 *JavaScript objects that represent the underlying REST API objects are referred to as JavaScript REST objects further on in this document.*

This section establishes the principle behind the Finesse JavaScript objects. For example, consider the Finesse REST API called User. A User is an agent who can log in to the Finesse Desktop with valid credentials. A User object can be a composition of various fields such as State, Dialogs, Phonebooks, and so on.

REST Collection Objects

Finesse JavaScript library provides REST collection objects, which is a collection of Finesse REST API objects. For instance, a User is a Finesse JavaScript REST object, and Users is a Finesse JavaScript REST collection object, which can hold multiple User objects.

Table 11: JavaScript REST Object with its Corresponding REST Collection Object

JavaScript REST Object	JavaScript REST Collection Object
Dialog	Dialogs
Media	MediaList
PhoneBook	PhoneBooks
Queue	Queues
Team	Teams
User	Users

Significance of REST Collection Object

To understand the significance of a REST collection object, consider the example of all the Dialogs associated with a User in a grid.

A Dialog is a JavaScript representation of a call. It can be a regular phone call, a conference, or a silent monitor session. A User object can be composed of many other objects, one of which is the Dialogs object. Note that it is Dialogs and not Dialog. This is because a user can be involved in multiple calls.

Example—Get Dialogs for a User

```
var _user = new finesse.restservices.User({
    id: '1001001', // user id
    onLoad: function(user) {

        // User has successfully loaded, now get User Dialogs
        user.getDialogs({
            onLoad: function(dialogs) {
               // successfully loaded the dialogs collection object, now format and display
 in the grid var dialogCollection = dialogs.getCollection();
                for (dialogId in dialogCollection) {
                    if (dialogCollection[dialogId] instanceOf finesse.restservices.Dialog)
 {
                       // each item in the Dialogs Collection will be an instance of Dialog
 object.
                        // can format and display each record here.
                    }
                }
            }
            // for the sake of simplicity of this example, not adding other handlers.
        });
    }
});
```

In the above example, after the User has successfully loaded, you call the `getDialogs()` API of the User object to get the Dialogs collection object and perform certain operations on it before you display it in a grid. Note that we did not explicitly initialize the Dialogs collection object. The `getDialogs()` API of the User object did it for us internally.

You do not have to explicitly create the collection objects. All the JavaScript REST objects which are composed of such collection object provides certain APIs which are internally taken care of initializing the objects. You must provide the handlers to control once the collection is loaded, modified, or deleted. The following are some examples of APIs which internally initialize and return respective REST collection objects.

Example—Collection Objects

```
_team.getUsers // Team REST API Object is composed of a Users Collection Object.(Users who
 are the part of that team)
_user.getMediaList // User REST API Object is composed of MediaList Collection Object.
_user.getQueues // User REST API Object is composed of Queues Collection Object.
_team.getTeamMessages // Team REST API Object is composed of a TeamMessages Collection
Object.
```

Consider the Dialogs in a grid example to include a feature where the grid updates in real-time if any new dialog shows up or any existing dialog gets removed. REST collection objects also provide multiple handlers. The following example shows two new handlers provided by the collection object `onCollectionAdd` and `onCollectionDelete` that are triggered when an item is added or removed respectively.

Example—Multiple Handlers

```
user.getDialogs({
    onLoad: function(dialogs) {},
    onCollectionAdd: function(dialog) {
        // called when a new Dialog is added to the collection
        // add this new dialog to the Grid
    },
    onCollectionDelete: function(dialog) {
        // called when a new Dialog is removed from the collection
        // remove this dialog from the Grid
    }
});
```

Commonly used REST collection object handlers are:

- onLoad—Triggers when the collection object is successfully loaded.
- onCollectionAdd—Triggers when a new item is added to the collection.
- onCollectionDelete—Triggers when a item is deleted from the collection.
- onError—Triggers when some error occurs during any of the above operations.

RestBase and RestCollectionBase Common Parameters

Finesse JavaScript library makes use of the principles of inheritance and composition extensively. To make the code more readable and maintainable, all the common functionality and properties are defined in Base classes. These Base classes are then extended by the child classes inheriting all their functionalities, overriding existing functionalities or adding new if needed.

All the JavaScript object classes such as User, Dialog, Media, Team, and Queue extend the RestBase class. All the REST collection object classes such as Users, Dialogs, MediaList, Teams, and Queues extend the RestCollectionBase class.

RestBase Common Parameters

All the JavaScript objects extend from the RestBase class. There is some common configuration that can be passed into each of these objects during initialization.

Example—Common Configurations

```
var _user = new finesse.restservices.User(options);
or
var _team = new finesse.restservices.Team(options);
or
var _media = new finesse.restservices.Media(options);
```

Example—Options

```
var options = {
    id: 'someUniqueId',
    onLoad: function(restObj){},
    onChange: function(restObj){},
    onAdd: function(restObj){},
    onDelete: function(restObj){},
    onError: function(response){},
}
```

Parameter	Description	Example
id	An ID that uniquely identifies the JavaScript object.	For a User A Each JavaSc of it on the

Parameter	Description	Example

onLoad	A callback that is invoked one time in the life of the object, which is when the initialization is successful and the data is loaded into the JavaScript object successfully. This JavaScript object is then passed to the handler as a parameter.
	
	This is equivalent to the success handler of a GET REST API request.
		An exan
		`var _us`
		`    id:`
		`    onL`
		`    }`
		`});`

onChange	A callback that is invoked upon the successful update of the object. The updated object is then passed to the handler as a parameter.
	
	This is equivalent to the success handler of a PUT REST API request.
		An exan
		`var _us`
		`    id:`
		`    onL`
		`    onC`
		`    }`
		`});`
		In the ab

onAdd	A callback that is invoked when an object is created in the upstream system by doing a POST request from the client. The client receives a success response for the creation. The newly created object is passed to the handler as a parameter. This is unlike the other scenarios where the JavaScript object is pre-existing in the system and you are creating a JavaScript version that creates a new object in the system.	Currentl
onDelete	A callback that is invoked when an object is deleted in the upstream system by doing a DELETE request from the client. The client receives a success response for the deletion.	Currentl

onError	A callback that is invoked with the response object as the parameter when any of the operations such as, GET, PUT, POST and DELETE fails.
	
	The following are the parameters:
	• status—`{Number}` Returns the HTTP status code.
	• content—`{String}` The raw string response.
	• object—`{Object}` The parsed object response.
	• error—The type of API error returned from the REST API request. It can be accessed using
	`rsp.object.ApiErrors.`
	`ApiError.ErrorType`
	  • errorType—`{String}` The type of error.
	  • errorMessage—`{String}` The message that is associated with the error.
		An exan
		`var _us`
		`    id:`
		`    onL`
		`    onE`
		`    }`
		`});`

RestCollectionBase Common Parameters

The RestCollectionBase objects are automatically created by the Finesse JavaScript Library when applicable APIs are used. Thus, the RestCollectionBase objects do not have to be initialized manually. The RestCollectionBase class extends the RestBase and supports all the handlers of RestBase.

The RestCollectionBase extends RestBase class. Hence, all the common parameters of the RestBase applies to RestCollectionBase.

Fields borrowed from class `finesse.restservices.RestBase`: ajaxRequestTimeout, restResponseStatus.

Methods borrowed from class `finesse.restservices.RestCollectionBase`: getCollection, refresh.

Methods borrowed from class `finesse.restservices.RestBase`: addHandler, getData, getId, getProperty, hasProperty, isLoaded, removeHandler.

Example of Common Configurations

```
_user.getDialogs(options) // User REST API Object is composed of Dialogs Collection Object.
_team.getUsers(options) // Team REST API Object is composed of a Users Collection
Object.(Users who are the part of that team)
_user.getMediaList(options) // User REST API Object is composed of MediaList Collection
Object.
_user.getQueues(options) // User REST API Object is composed of Queues Collection Object.
_team.getTeamMessages(options) // Team REST API Object is composed of a TeamMessages
Collection Object.
```

Parameter	Description
onLoad	A callback that is invoked one time in the life of the object, which is when the initializati successful and the data is loaded into the JavaScript object successfully. This JavaScript c then passed to the handler as a parameter. This is equivalent to the success handler of a GET REST API request.
onCollectionAdd	A callback that is invoked when a new object is added to the collection. The newly added then passed to the handler as a parameter.
onCollectionDelete	A callback that is invoked when an object is removed from the collection. The removed c then passed to the handler as a parameter.

Parameter	Description
onError	A callback that is invoked with the response object as the parameter when any of the such as, GET, PUT, POST and DELETE fails. For more information on parameters, see RestBase Common Parameters, on page 496

JavaScript Library

Without Finesse JavaScript library

In the absence of the Finesse JavaScript library, the following code would be needed to pull the User details after the login process is completed.

- Make a GET call to the server to get the details of this agent.

Example—GET

```
$.ajax({
    url: 'finesse/api/User/1001001' // where 1001001 is the username or id of the logged
 in agent.
    type: 'GET',
    success: function(response){
        // here response is an xml containing all the relevant information regarding
the User 1001001 which can be used
        // for example response can be
<User><id>1001001</id><state>NOT_READY</state></User>
        // Need to parse the response and make it usable
    },
    error: function(err){
        // Something went wrong while fetching user data, show an error dialog to the
user may be.
    },
});
```

- Make a PUT call to the server to change the state of this agent.

Example—PUT

```
$.ajax({
    url: 'finesse/api/User/1001001' // where 1001001 is the username/id through which
the agent logged in.
    type: 'PUT',
    data: '<User><state>READY</state></User>'
    success: function(response){
        // Do something once the user state has been changed successfully.
    },
    error: function(err){
        // Something went wrong while fetching user data, show an error dialog to the
user may be.
    },
});
```

REST operations such as POST and DELETE can also be performed on the User API to get the desired result.

With Finesse JavaScript library

In the presence of the Finesse JavaScript library, the following code would be needed to pull the User details, where the User object is under the namespace `finesse.restservices.User`.

- Make a GET all on the User API. The GET call in the above example is made using the jQuery OpenAjax API, where the `onLoad` is equivalent to the `success` option of the jQuery OpenAjax call.

  Example—onLoad, onChange, and on Error

  ```
  var _user = new finesse.restservices.User({
      id: '1001001',
      onLoad: function(user){
          // Do something on the successful fetch(GET) of user object
      },
      onChange: function(user){
          // Do something on the successful update(PUT) of object
          // can do user.getState() or user.getTeamName()
      }
      onError: function(err){
          // Something went wrong while fetching user data, show an error dialog to the
  user may be.
      }
  });
  ```

  The `onChange` is equivalent to the `success` option in PUT jQuery OpenAjax call made to modify the state of the User in the second example. Similarly, there are other handlers such as `onAdd` used for POST request and `onDelete` used for DELETE requests which are supported by User object as well as other Finesse JavaScript objects.

- Update the state of the User using the `setState()` provided by finesse.restservices.User.

  Example—Update State of a User

  ```
  _user.setState('READY');
  ```

  The above example triggers a state change for the User, which is equivalent to make a PUT request, which in turn triggers the `onChange` handler attached to the User object.

  All the handlers (GET, PUT, POST, DELETE, ERROR) can be attached to the object during initialization. Initialization of a JavaScript object triggers a GET request, the response of which is used to populate the JavaScript object. There are APIs available within the JavaScript object to create, update, and delete certain compositions (in the JavaScript object itself) that internally trigger PUT, POST, and DELETE REST API request respectively.

  To put this into perspective, the Finesse JavaScript REST API objects try to encapsulate the low-level request or response handling at the client-side and provide with APIs which are easy to use, maintain and improve the readability of the code.

Subscription Support

Finesse JavaScript objects support subscription to the XMPP events. These events are the notifications generated by the Openfire server and pushed to the desktop as XMPP events. Any JavaScript object that supports subscription is automatically hooked up for listening to XMPP events when it is initialized.

Note This subscription is not a Notification subscription but a desktop level subscription to receive the events generated by the OpenAjax hub.

For example, the User API supports the subscription model and the User functions as stated in the below example.

Example for User Functions

```
var _user = new finesse.restservices.User({
    id: '1001001',
    onLoad: function(user){
        // Do something on the successful fetch(GET) of user object
    },
    onChange: function(user){
        // Do something on the successful update(PUT) of object
        // can do user.getState() or user.getTeamName()
    },
    onAdd: function(user){},
    onDelete: function(user){},
    onError: function(err){
        // Something went wrong while fetching user data, show an error dialog to the user
 may be.
    }
});
```

These handlers can handle the REST response and the XMPP events. For example, when the state change for a signed-in User is triggered by another agent (that is, Supervisor), the client or desktop receives a User update XMPP event on the node “/finesse/api/User/1001001”.

Payload with Updated User State Details

```
<Update>
  <data>
    <user>
      <dialogs>/finesse/api/User/1001001/Dialogs</dialogs>
      <extension>1001001</extension>
      <firstName>AGENT</firstName>
      <lastName>1001001</lastName>
      <loginId>1001001</loginId>
      <loginName>agent444agent444agent444agent444</loginName>
      <mediaType>1</mediaType>
      <pendingState></pendingState>
      <roles>
        <role>Agent</role>
      </roles>
      <settings>
        <wrapUpOnIncoming>OPTIONAL</wrapUpOnIncoming>
        <wrapUpOnOutgoing>NOT_ALLOWED</wrapUpOnOutgoing>
      </settings>
      <state>READY</state>
      <stateChangeTime>2020-03-13T05:45:26.827Z</stateChangeTime>
      <teamId>5000</teamId>
      <teamName>FunctionalAgents</teamName>
      <uri>/finesse/api/User/1001001</uri>
      <wrapUpTimer>30</wrapUpTimer>
    </user>
  </data>
  <event>PUT</event>
  <requestId>2a7f6cd3-bd26-4e46-a8ea-429cba8d9ff7</requestId>
  <source>/finesse/api/User/1001004</source>
</Update>
```

The XMPP events are handled, and the same `onChange` handler provided by you is invoked by the Finesse JavaScript library.

REST Base

Class finesse.restservices.RestBase

Represents the JavaScript REST object and it exposes methods to operate on the object against the server. This object is extended to individual JavaScript REST objects (such as Dialog, User, and so on) and is not be used directly.

Example

```
var NotReadyReasonCode = RestBase.extend( /** @lends
finesse.restservices.NotReadyReasonCode.prototype */ {

            /**
             * @class
             * A ReasonCode object for NOT_READY state.
             *
             * @augments finesse.restservices.RestBase
             * @see finesse.restservices.User.States#NOT_READY
             * @constructs
             */
            _fakeConstuctor: function() {
                /* This is here to hide the real init constructor from the public docs */
            },
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Field Details

Name	Description
ajaxRequestTimeout	Duration in milliseconds that the OpenAjax request remains opened.
restResponseStatus	Number of the REST response status that is returned.

Methods

addHandler(notifierType, callback, scope)

Add a handler for the specific RestBase object. The callback function is invoked if the notifierType is triggered.

Example

```
// Handler for additions to the Dialogs collection object.
// When Dialog (a RestBase object) is created, add a change handler.
_handleDialogAdd = function(dialog) {
    dialog.addHandler('change', _handleDialogChange);
}
```

Parameters

Name	Type	Description	Required
notifierType	String	The type of notifier to add to the load, change, add, delete, and error.	Yes
callback	Function	An asynchronous callback function that is invoked when the type is notified.	Yes

Name	Type	Description	Required
scope	Object	The object on which the handler is invoked.	Optional

getData()

Retrieves the data for an object.

Returns

`{Object}` The object with the retrieved data.

getId()

Retrieves the unique identifier of the RestBase.

Returns

`{String}` Unique identifier of the RestBase.

getProperty(obj, property)

Retrieves the property from the object.

Parameters

Name	Type	Description	Required
obj	Object	The object to retrieve the property from.	Yes
property	String	The property is the key of the value that will be returned.	Yes

Returns

`{Object}` The value of the property that was requested.

hasProperty(obj, property)

Determines whether the object has a property.

Parameters

Name	Type	Description	Required
obj	Object	The object to check if the property exists.	Yes
property	String	The property is the key of the value that will be returned.	Yes

Returns

`{Boolean}` True if the object contains the property, else false.

isLoaded()

Loads the utility method for operations that require complete instantiation.

Throws

`{Error}` If this object is not fully instantiated.

Returns

`{finesse.restservices.RestBase}` The RestBase object reference. Makes the isLoaded function available to all the classes which are extending RestBase class. For example, in Dialogs, this.isloaded() can be called Dialogs.js which is the child class of RestBase.

refresh(retries)

Updates the RestBase object with the latest data by performing an asynchronous GET. The updated object will be returned through the `onChange` handler, so make sure it is registered.

Parameters

Name	Type	Description	Required
retries	Integer	The number of retry attempts to update the RestBase object.	Yes

Returns

`{Object}` The end-call function that signifies the callback handler to not process the response of the asynchronous request.

removeHandler(notifierType, callback)

Removes previously added handler for the specified notifierType.

Parameters

Name	Type	Description	Required
notifierType	String	The type of notifier to remove the load, change, add, delete, and error.	Yes
callback	Function	The callback to be removed.	Yes

REST Collection Base

Class finesse.restservices.RestCollectionBase

Extends finesse.restservices.RestBase Common Parameters

Represents the collection of `finesse.restservices.RestBase` objects. A collection is a group of similar objects. For instance, Users is a collection that can hold multiple User objects.

This class is used by all other JavaScript objects. For more information, see RestCollectionBase Common Parameters, on page 498.

Methods

getCollection()

Retrieves the RestBase collection.

Returns

`{Object}` The collection as an object.

refresh()

Updates the RestBase object with the latest data by performing an asynchronous GET. The updated object will be returned through the `onChange` handler, so make sure it is registered

Returns

`{finesse.restservices.RestBaseCollection}` The RestBaseCollection object reference. Makes the refresh function available to all the classes which are extending RestCollectionBase class.

User

Class finesse.restservices.User

Extends finesse.restservices. RestBase Common Parameters

Represents an agent or supervisor and includes information about the user, such as roles, state, teams, dialogs, and so on. The User object is the representation of the Finesse REST API User object.

When the User object is initialized (for example, `var _user = new finesse.restservices.User()`), a GET REST API request is made to `/finesse/api/User/<userid>`, and its response is used to populate the User object.

When a User change event is received, the User object's values are updated accordingly. For example, if the agent state changes, the respective User object's `getState()` method reflects the change, and returns the latest state of the agent when invoked.

Example

```
var _user = new finesse.restservices.User({
    id: _id,
    onLoad: _handleUserLoad,
    onChange: _handleUserChange
});
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Methods

getActiveDeviceId()

Retrieves the current active device ID of the agent.

Example

```
var _user = new finesse.restservices.User({
    id: _id,
    onLoad: _handleUserLoad,
    onChange: _handleUserChange
});
_user.getActiveDeviceId();
```

Returns

`{String}` The active device ID for that agent.

getDevices()

Retrieves the list of devices associated with a particular extension.

Example

```
var _user = new finesse.restservices.User({
    id: _id,
    onLoad: _handleUserLoad,
    onChange: _handleUserChange
});
_user.getDevices();
```

Returns

`{Object}` The collection object of the devices that is associated with a particular extension. The contents of a device include the following:

- deviceId—A unique ID of the device.
- deviceType—The device type as defined in the CiscoTerminal.getType() in JTAPI specifications.
- deviceTypeName—The display name of the device type as defined in the CiscoTerminal.getTypeName() in JTAPI specifications.

For more information about JTAPI specifications, refer to Cisco Unified JTAPI Developers Guide.

getDialogs(handlers)

Retrieves the collection of voice dialogs associated with the current user. This includes the dialogs that the user is currently active on, being alerted, along with the held dialogs. The terminated dialogs are not part of the list.

The dialog list is retrieved by making a GET REST API request to the `/finesse/api/User/<id>/Dialogs/` endpoint. The `getDialogs` are queried only once from the server that is, when the object is created.

Example

```
_dialogs = _user.getDialogs({
    onCollectionAdd: handleNewDialog,
    onCollectionDelete: handleEndDialog
});
```

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered. To find the list of callback scenarios, see RestCollectionBase Common Parameters, on page 498.	Optional

Returns

`{finesse.restservices.Dialogs}` The Dialogs collection object.

getDialogsNoCache(handlers)

Retrieves the collection of dialogs (calls) associated with the current user. This includes the dialogs that the user is currently active on, being alerted, along with the held dialogs. The terminated dialogs are not part of the list. The difference between the `getDialogsNoCache` and `getDialogs` methods is that the GET REST API request is always made for this method.

Example

```
_user.getDialogsNoCache({
    onLoad: handleDialogsLoadedCallDetails,
    onCollectionAdd: handleDialogsAddedCallDetails,
    onCollectionDelete: handleDialogsDeletedCallDetails,
    onError: handleDialogsErrorCallDetails
});
```

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered.	Optional

Returns

`{finesse.restservices.Dialogs}` The Dialogs collection object.

getExtension()

Retrieves the extension that is associated with the user.

Returns

`{String}` The extension of the user.

getFirstName()

Retrieves the first name of the user.

Returns

`{String}` The first name of the user.

getFullName()

Retrieves the full name of the user. The full name format is FirstName LastName (for example, John Doe).

Returns

`{String}` The full name of the user.

getLastName()

Retrieves the last name of the user.

Returns

`{String}` The last name of the user.

getMediaList(handlers)

Retrieves the media list that is associated with the user. It retrieves the media dialog collection object.

Example

```
var mediaList = _user.getMediaList({
    onCollectionAdd: _handleMediaAdd,
    onCollectionDelete: _handleMediaDelete,
    onLoad: _handleMediaListLoaded
});
```

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered. To find the list of callback scenarios, see RestCollectionBase Common Parameters, on page 498.	Optional

Returns

`{finesse.restservices.MediaList}` The MediaList collection object.

getMediaPropertiesLayout(handlers)

Retrieves the layout that is associated with the user. Teams are configured with custom layouts by the administrator. Users are associated to custom call variable layouts (MediaPropertyLayout) due to their association with a team.

Example

```
var _mediaPropertiesLayout = _user.getMediaPropertiesLayout({
    onLoad: _handleMediaPropertiesLayoutLoaded,
    onError: _handleMediaPropertiesLayoutError
});
```

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered. To find the list of callback scenarios, see RestBase Common Parameters, on page 496.	Optional

Returns

`{finesse.restservices.UserMediaPropertiesLayout}` The UserMediaPropertiesLayout object.

getMediaPropertiesLayouts(handlers)

Retrieves the layouts that is associated with the user. Teams are configured with custom layouts by the administrator. Users are associated to custom call variable layouts (MediaPropertyLayouts) due to their association with a team.

Example

```
var _mediaPropertiesLayouts = _user.getMediaPropertiesLayouts({
    onLoad: _handleMediaPropertiesLayoutsLoaded,
    onError: _handleMediaPropertiesLayoutsError
});
```

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered. To find the list of callback scenarios, see RestCollectionBase Common Parameters, on page 498.	Optional

Returns

`{finesse.restservices.UserMediaPropertiesLayout}` The UserMediaPropertiesLayout object.

getMediaState()

Retrieves the media state of the user. This is applicable only in Unified CCX deployments.

Returns

`{String}` The current (or last fetched) media state of the user. When the agent is talking on a manual outbound call, or when the agent in not signed-in, the getMediaState returns busy. In all other cases getMediaState returns null.

getMobileAgentDialNumber()

Retrieves the mobile agent dial number.

Returns

`{String}` If available, returns the mobile agent dial number, otherwise null.

getMobileAgentMode()

Retrieves the mobile agent work mode. In Unified CCE, when an agent has logged in as a mobile agent (by selecting **Sign in as a Mobile Agent** checkbox in the Cisco Finesse login page), then it returns mobile agent mode. If an agent has not selected the checkbox, then it returns null.

Returns

`{finesse.restservices.User.WorkMode}` The WorkMode object. If available, then the mobile agent work mode, otherwise null. For more information, see User.WorkMode, on page 521.

getNotReadyReasonCodeId()

Retrieves the user's Not Ready reasonCodeId.

Returns

`{String}` The reasonCodeId, or undefined if the ID is not set or indeterminate.

getNotReadyReasonCodes(handlers)

Retrieves all the custom Not Ready reason codes configured globally and the team level reason codes applicable to the user.

Note There is no return value. Use the success handler to process a valid return.

```
_user.getNotReadyReasonCodes({
    success: handleNotReadyReasonCodesSuccess,
    error: handleNotReadyReasonCodesError
});
```

Parameters

Name	Type	Description	Required
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Optional

getPendingStateReasonCode()

Retrieves the pending state reasonCode of the user.

Returns

`{Object}` The reasonCode for the pending state of the user. The contents include the following:

- uri—The URI for the reasonCode object.
- Id—The unique ID for the reasonCode.
- category—The category can either be NOT_READY or LOGOUT.
- code—The numeric reasonCode value.
- label—The label for the reasonCode.
- forAll—The boolean flag that denotes the global status for the reasonCode.
- systemCode—The boolean flag which denotes whether the reasonCode is system-generated or customized.

getPhoneBooks(handlers)

Retrieves the PhoneBooks collection object that is associated with the user.

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered. To find the list of callback scenarios, see RestCollectionBase Common Parameters, on page 498.	Optional

Returns

`{finesse.restservices.PhoneBooks}` The PhoneBooks collection object.

getQueues(handlers)

Retrieves the Queues collection object that is associated with the user.

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered. To find the list of callback scenarios, see RestCollectionBase Common Parameters, on page 498.	Optional

Returns

`{finesse.restservices.Queues}` The Queues collection object.

getReasonCode()

Retrieves the reason code object corresponding to the user's current state.

Returns

`{Object}` The reasonCode for the pending state of the user. The contents include the following:

- uri—The URI for the reasonCode object.
- id—The unique ID for the reasonCode.
- category—The category and it can be either NOT_READY or LOGOUT.
- code—The numeric reasonCode value.
- label—The label for the reasonCode.
- forAll—Boolean flag that denotes the global status for the reasonCode.
- systemCode—Boolean flag which denotes whether the reasonCode is system-generated or customized.

getReasonCodeById(handlers, reasonCodeId)

Retrieves the reason code object that is associated with the reasonCodeId.

Note There is no return value. Use the success handler to process a valid return.

Parameters

Name	Type	Description	Required
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes
reasonCodeId	String	Unique identifier for the reasonCode to lookup.	Yes

getReasonCodeLabel()

Retrieves the user's reasonCode label for both Not Ready and Logout reasonCodes.

Returns

`{String}` The reasonCode label, or an empty string if none.

getServices()

Retrieves the list of services configured for the user.

The following service can be configured for the user.

Agent Answers: A real time presentation of suggestions for the agent to consider based on the live conversation between the end customer and agent.

This is applicable only in Unified CCE deployments.

Example

```
var services = user.getServices();
```

Returns

`{Object}` The array list of services.

getSignoutReasonCodes(handlers)

Retrieves all the Signout reason codes that is associated with the user.

Note There is no return value. Use the success handler to process a valid return.

```
_user.getSignoutReasonCodes({
    success: handleSignoutReasonCodesSuccess,
    error: handleSignoutReasonCodesError
});
```

Parameters

Name	Type	Description	Required
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

getSkillTargetId()

Retrieves the ID for the skill target assigned to the user in the Unified CCE database. It is supported from Cisco Finesse, Release 12.5(1) ES2 onwards.

Note This is only supported for Unified CCE deployments.

Returns

`{String}` The ID for the skill target assigned to the user.

getState()

Retrieve the state of the user.

Returns

`{String}` The current (or last fetched) state of the user.

getStateChangeTime()

Retrieves the state change time (UTC) of the user.

Returns

`{String}` The state change time of the user in UTC.

getSupervisedTeams()

Retrieves the teams that are managed by the user (supervisor). Applicable for users that are supervisors.

Returns

`{Array}` The array of objects containing Id, name, and URI of the teams managed by the user (supervisor).

The object content includes the following:

- id—The unique ID for the team.
- name—The team name for the team.
- uri—The URI for the team.

getTeamId()

Retrieves the ID of the team that is associated with the user.

Returns

`{String}` The current (or last fetched) ID of the team that is associated with the user.

getTeamName()

Retrieves the name of the team that is associated with the user.

Returns

`{String}` The current (or last fetched) name of the team that is associated with the user.

getWrapUpOnIncoming()

Retrieves the wrap-up mode of the user. For more information, see User.WrapUpMode, on page 521.

Returns

`{String}` The wrap-up mode of the user.

getWrapUpOnOutgoing()

Retrieves the wrap-up mode of the user. For more information, see User.WrapUpMode, on page 521.

Returns

`{String}` The wrap-up mode of the user.

getWrapUpReasons(handlers)

Retrieves the WrapUpReasons collection object that is associated with the user.

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered. To find the list of callback scenarios, see RestCollectionBase Common Parameters, on page 498.	Optional

Returns

`{finesse.restservices.WrapUpReasons}` The WrapUpReasons collection object.

getWrapUpTimer()

Retrieves the maximum amount of time the user can be in Wrap Up state (in seconds).

Returns

`{String}` The WrapUp time configured for the user. For example, 3600 (1 hour).

hasAgentRole()

Checks whether the user is an agent.

Returns

`{Boolean}` True if the user has the role of an agent, else false.

hasSupervisorRole()

Checks whether the user is a supervisor.

Returns

`{Boolean}` True if the user has the role of the supervisor, else false.

isDeviceSelectionEnabled()

Retrieves whether the device selection is enabled for the user.

Example

```
/**
  * Retrieves whether the device selection is enabled for the user.
  * If device selection is enabled, the login request for the user should
  * contain active device Id, if the extension chosen by the agent is
  * shared between multiple devices.
  * @see finesse.restservices.User.loginWithActiveDeviceId
  * @returns {Boolean} True if the device selection is enabled and false
  *          if device selection is disabled.
  */
 isDeviceSelectionEnabled: function() {
     this.isLoaded();
     if (this.getData().settings) {
         return this.getData().settings.deviceSelection === 'enabled';
     }
     return false;
 }
```

Returns {Boolean} True if the device selection is enabled and false if device selection is disabled.

isMobileAgent()

Checks whether the user is a mobile agent.

Returns

`{Boolean}` True if this agent is a mobile agent, else false.

isPendingStateChange()

Checks whether there is a pending state change. A pending state change is a request to change state that does not result in an immediate state change. For example, if an agent in the TALKING state attempts to change to the NOT_READY state, the state is not changed until the call ends. Pending state change occurs when the agent is in the following states:

- TALKING
- HOLD
- RESERVED
- OUTBOUND
- PREVIEW

Returns

`{Boolean}` True if there is a pending state change.

isReasonCodeReserved()

Checks whether the reasonCode of the user is a system-generated reasonCode.

Returns

`{Boolean}` True if the reasonCode for the state of the user is a system-generated reasonCode.

isWrapUp()

Checks whether the user's current state is WORK or WORK_READY. This is used to ensure that a pending state is not cleared when moving into wrap-up (work) mode. We do not add this as a pending state, as the changes (while in wrap-up) occur immediately.

Returns

`{Boolean}` True if user is in wrap-up mode.

isWrapUpRequired()

Checks whether the user is required to go into wrap-up mode.

Returns

`{Boolean}` True if the user is required to go in to wrap-up mode.

login(extension, handlers)

Performs an agent login for the user and associates the agent with the specified extension.

Parameters

Name	Type	Description	Required
extension	String	The extension to associate with the user.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.User}` The User object.

loginMobileAgent(extension, mode, extension, handlers, reasonCode)

Performs an agent login for the user and associates the agent with the specified extension. This marks the agent as a mobile agent and associates an external dial number.

Parameters

Name	Type	Description	Required
extension	String	The extension to associate with the user.	Yes
mode	String	The mobile agent work mode as defined in `finesse.restservices.User.WorkMode`. For more information, see User.WorkMode, on page 521.	Yes
extension	String	The external dial number to be used by the mobile agent.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Name	Type	Description	Required
reasonCode	Object	An object containing the reasonCode for the login request.	Yes

Returns

`{finesse.restservices.User}` The User object.

loginWithActiveDeviceId(extension, activeDeviceId, handlers)

Performs an agent login for a user and associates the agent with the specified extension and device.

Example

```
var _user = new finesse.restservices.User({
    id: _id,
    onLoad: _handleUserLoad,
    onChange: _handleUserChange
});
_user.loginWithActiveDeviceId(extension, activeDeviceId, handlers);


var handlers = {
     success: function() {
        // handler for success response
     },
     error: function() {
        //handler for error response
     }
}
```

Parameters

Name	Type	Description	Required
extension	String	The extension to associate with the user.	Yes
activeDeviceId	String	The device ID to associate with the user.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers.	Yes

Returns

`{finesse.restservices.User}` The User object.

logout(reasonCode, handlers)

Performs an agent logout for the user.

Parameters

Name	Type	Description	Required
reasonCode	String	The reason that the user is logging out.	Yes

Name	Type	Description	Required
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.User}` The User object.

makeBargeCall(number, dialogUri, handlers)

Makes a silent monitor call to a particular agent's phone number. Barge in to call, which is silently monitored by the supervisor.

Note Applicable for users that are supervisors. Barge in is performed on a call that is not being monitored by the supervisor, and the error handler is invoked.

Parameters

Name	Type	Description	Required
number	String	The agent's extension of the call that is being barged into.	Yes
dialogUri	String	The associated dialog URI of SUPERVISOR_MONITOR call.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.User}` The User object.

makeCall(number, handlers)

Makes a call to the specified phone number.

Parameters

Name	Type	Description	Required
number	String	The phone number to call.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.User}` The User object.

makeSMCall(number, handlers)

Makes a silent monitor call for the specified agent's extension.

Parameters

Name	Type	Description	Required
number	String	The phone number to silent monitor.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.User}` The User object.

setState(newState, reasonCode, handlers)

Sets the state of the user.

Parameters

Name	Type	Description	Required
newState	String	The state that you are setting for the user.	Yes
reasonCode	ReasonCode	The reason that the user is logging out.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.User}` The User object.

updateToMobileAgent(mode, dialNumber, handlers)

Updates the user object with the agent's mobile login information.

Parameters

Name	Type	Description	Required
mode	String	The mobile agent work mode as defined in finesse.restservices.User.WorkMode. For more information, see User.WorkMode, on page 521.	Yes
dialNumber	String	The phone number that is used by the mobile agent.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.User}` The User object.

User.MediaStates

Class finesse.restservices.User.MediaStates

When the agent is talking on a manual outbound call, the media state returns BUSY. Otherwise, it returns null.

Field Details

Name	Description
BUSY	The user is on a manual outbound call. This is applicable only for Unified CCX deployments.

User.States

Class finesse.restservices.User.States

Represents the possible user state values.

Field Details

Name	Description
HOLD	The user is on hold. In Unified CCX deployments, the user remains in TALKING state while on hold.
LOGIN	The user logs in. **Note** This is an action, not a state since a logged-in user is always in a specific state (READY, NOT_READY, TALKING and so on).
LOGOUT	The user has logged out.
NOT_READY	The user is not ready. In Unified CCX deployments, the user is in this state while on a non-routed call.
READY	The user is ready for calls.
RESERVED	The user has a incoming call, but has not answered it.
RESERVED_OUTBOUND	The user has an outbound call, but not connected to it.
RESERVED_OUTBOUND_PREVIEW	The user has an outbound call's preview information, but has not acted on it.
TALKING	The user is on a call. In Unified CCX deployments, this is for routed calls only.

Name	Description
WORK	The user is in wrap-up or work mode. This mode is configured to time out. After time out, the user's state changes to NOT_READY.
WORK_READY	The user is in wrap-up or work mode. This mode is configured to time out. After time out, the user's state changes to READY.

User.WorkMode

Class finesse.restservices.User.WorkMode

WorkMode is only applicable for Unified CCE and mobile agents. When an agent has logged in as a mobile agent (by selecting **Sign in as a Mobile Agent** checkbox in the Cisco Finesse login page), then the agent must select the work mode from the drop-down list. The following are the work modes:

- **Call by Call**
- **Nailed Collection**

Field Details

Name	Description
CALL_BY_CALL	The mobile agent is connected (dialed) for each incoming call received, and is disconnected when the call ends.
NAILED_CONNECTION	The mobile agent is connected (dialed) at login and the call stays connected through multiple customer calls.

User.WrapUpMode

Class finesse.restservices.User.WrapUpMode

Represents the possible wrap-up mode types in Unified CCE deployments.

Field Details

Name	Description
NOT_ALLOWED	The user is not allowed to go to wrap-up when call ends.
OPTIONAL	The user can choose to go to wrap-up on a call-by-call basis when the call ends.
REQUIRED	The user must go to wrap-up when call ends.
REQUIRED_WITH_WRAP_UP_DATA	The user must go to wrap-up when call ends and must enter wrap-up data.

UserMediaPropertiesLayout

Class finesse.restservices.UserMediaPropertiesLayout

Represents a media properties layout.

Example

```
_mediaPropertiesLayout = _user.getMediaPropertiesLayout({
    onLoad: _handleMediaPropertiesLayoutLoaded,
    onError: _handleMediaPropertiesLayoutError
});
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Method

getRestUrl()

Retrieves the URL for the UserMediaPropertiesLayout resource.

UserMediaPropertiesLayouts

Class finesse.restservices.UserMediaPropertiesLayouts

Represents a collection of media properties layouts.

Example

```
var _mediaPropertiesLayouts = _user.getMediaPropertiesLayouts({
    onLoad: function(mediaPropertiesLayouts) {},
    onError: function(error) {}
});
```

Parameters

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

Users

Class finesse.restservices.Users

Extends finesse.restservices.RestCollectionBase

Represents a collection of User objects. When there is no method to retrieve all Users, then this collection is used to return Users in a team.

Example

```
// Note: The following method gets an Array of Teams, not a Collection.
_teams = _user.getSupervisedTeams();
if (_teams.length > 0) {
    _team0Users = _teams[0].getUsers();
}
```

Parameters

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

Dialog

Class finesse.restservices.Dialog

Extends finesse.restservices.DialogBase

Represents a call such as a regular phone call, a conference, or a silent monitor session.

Example

```
var _dialogs = _user.getDialogs({
        includeNonVoice: true
    }
    _dialogs.addHandler('load', function() {
        dialog
    }) dialogCollection = _dialogs.getCollection();
    for (dialog in dialogCollection) {
        if (dialogCollection.hasOwnProperty(dialog)) {
            var _dialog = dialogCollection[dialog];
            _dialog.addHandler('change', function() {
                // TODO: on change of dialog do some thing
            }));
    }
}
```

Methods

cancelCallback(mediaAddress, handlers)

Cancels a scheduled callback.

Parameters

Name	Type	Description	Required
mediaAddress	String	The media address of the user.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

dropParticipant(targetMediaAddress, handlers)

Drops a participant from the call.

Parameters

Name	Type	Description	Required
targetMediaAddress	String	The media address of the participant to drop from the call.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

getCallbackInfo()

Retrieves information about the currently scheduled callback.

Returns

`{Object}` Undefined if no callback is set. If callback is set, then it returns a map with one or more entries depending on the configured value. The callbackTime refers to the configured callback time and the callbackNumber refers to the configured callback number.

getCallbackNumber()

Retrieves the callback number without the dialer prefix. This is required to schedule a callback if there is any dialer prefix added for Direct Preview Outbound calls.

Returns

`{String}` The callback number without the dialer prefix.

getDroppableParticipants(filterExtension)

Retrieves the dropable participants, which are participants with an agent extension. A participant can represent an internal user (such as, an agent) or an external user (such as, a customer). It is not a CTI Route Point, IVR Port, or the caller.

Parameters

Name	Type	Description	Required
filterExtension	String	To remove a single extension from the list.	Optional

Returns

`{Array}` The array of participants that can be dropped. Participant entity properties are as follows:

- state—The last participant state in a dialog.
- stateCause—The reason for the last participant state in a dialog. It is applicable to a FAILED participant state. The possible values are BUSY, BAD_DESTINATION, and OTHER.
- mediaAddress—The media address of the participant.
- startTime—The start time of the participant.
- stateChangeTime—The time from when the participant state has changed.
- actions—The list of actions that a participant can perform.

getFromAddress()

Retrieves the from address, which is the calling line ID of the caller.

Returns

`{String}` The from address.

getParticipantTimerCounters(participantExt)

Retrieves the participant timer counters.

Parameters

Name	Type	Description	Required
participantExt	String	The extension of the participant.	Yes

Returns

`{Object}` The array of participants which contains properties:

- state—The last participant state in a dialog.
- startTime—The start time of the participant.
- stateChangeTime—The time when the participant state has changed.

getSecondaryId()

Retrieves the secondaryId of a dialog. A consult call has two call legs (primary leg and a consult leg). When the consult call is completed (either with a transfer or conference), the two call legs would merge. The dropped call's Dialog Id can be found in the secondaryId field of the the surviving call's Dialog object. For Unified CCE deployments, the secondaryId will be populated for direct transfers as well. This is supported from Cisco Finesse, Release 11.6(1) ES1 onwards.

Returns

`{String}` The Id of the secondary dialog.

getServices()

Retrieves the list of services available for this dialog. Services are only available in Unified CCE deployments.

Example

```
var services = dialog.getServices();
```

Returns

`{Object}` The array list of services.

getToAddress()

Retrieves the to address, which is the destination for the call.

Returns

`{String}` The to address.

initiateDirectTransfer(mediaAddress, toAddress, handlers)

Initiates a single-step transfer request.

Parameters

Name	Type	Description	Required
mediaAddress	String	The extension of the user performing the single-step transfer.	Yes
toAddress	String	The destination address of the single-step transfer.	Yes

Name	Type	Description	Required
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

isParticipantDroppable(participantExt)

Determines whether the supervisor can drop the provided extension from a call.

Parameter

Name	Type	Description	Required
participantExt	String	The extension of the participant.	Yes

Returns

`{Boolean}` True if the extension of the participant is droppable.

makeConsultCall(mediaAddress, toAddress, handlers)

Makes a consult call to a destination.

Parameters

Name	Type	Description	Required
mediaAddress	String	The extension of the user performing the consult call.	Yes
toAddress	String	The destination address of the consult call.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

reclassifyCall(mediaAddress, classification, handlers)

Reclassifies an Outbound Option Direct Preview call. A call can be reclassified as VOICE, FAX, ANS_MACHINE, INVALID, DO_NOT_CALL, or BUSY. The call type is then sent back to Unified CCX for processing.

Parameters

Name	Type	Description	Required
mediaAddress	String	The extension of the user performing the reclassification.	Yes
classification	String	The new classification to assign to the call. Valid values are VOICE, FAX, ANS_MACHINE, INVALID, BUSY, and DO_NOT_CALL.	Yes

Name	Type	Description	Required
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

requestAction(mediaAddress, action, handlers)

Requests the provided action to be taken for the specified mediaAddress.

Example

```
dialog.requestAction(_user.getExtension(), 'CONSULT', {
    error: onError,
    success: onSuccess
});
```

Parameters

Name	Type	Description	Required
mediaAddress	String	The extension of the user which the action is performed on.	Yes
action	String	The action that is invoked on the dialog. For more information, see Dialog.Actions, on page 528.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

sendDTMFRequest(mediaAddress, handlers, digit)

Sends the DTMF digit tones.

Parameters

Name	Type	Description	Required
mediaAddress	String	The extension of the user sending the DTMF digits.	Yes
digit	String	The DTMF digits to be sent.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

updateCallbackNumber(mediaAddress, callbackNumber, handlers)

Updates the number for a callback.

Parameters

Name	Type	Description	Required
mediaAddress	String	The extension of the user updating the callback number.	Yes
callbackNumber	String	The new number for the callback.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

updateCallbackTime(mediaAddress, callbackTime, handlers)

Updates the time for a callback.

Parameters

Name	Type	Description	Required
mediaAddress	String	The extension of the user updating the callback time.	Yes
callbackTime	String	The new time for the callback. Format: YYYY-MM-DDTHH:MM For example, 2013-12-24T23:59	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

updateWrapUpReason(wrapUpReason, handlers)

Updates the wrap-up reason for the dialog.

Parameters

Name	Type	Description	Required
wrapUpReason	String	The new wrap-up reason for the dialog.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Dialog.Actions

Class finesse.restservices.Dialog.Actions

The list of actions that can be taken on a dialog.

Field Details

Name	Description
ACCEPT	The user accepts a dialog that is being previewed.
ANSWER	The user answers a dialog.
BARGE_CALL	The user barges into a dialog.
CANCEL_SCHEDULED_CALLBACK	The user cancels a scheduled callback.
CLOSE	The user closes a dialog.
CONFERENCE	The user initiates a conference of a dialog.
CONSULT_CALL	The user initiates a consult call.
DROP	The user drops from the dialog.
DTMF	The user sends DTMF digits to a dialog.
HOLD	The user puts the dialog on hold.
MAKE_CALL	The user makes a new dialog.
PARTICIPANT_DROP	The supervisor drops a participant from the dialog.
RECLASSIFY	The user changes the classification for the dialog.
REJECT	The user rejects a dialog.
RETRIEVE	The user retrieves a dialog that is on Hold.
START_RECORDING	The user initiates a recording on a dialog.
TRANSFER	The user initiates a transfer of a dialog.
TRANSFER_SST	The user initiates a single-step transfer of a dialog.
UPDATE_CALL_DATA	The user updates data on a dialog.
UPDATE_SCHEDULED_CALLBACK	The user updates the time or number for a scheduled callback.

Dialog.ParticipantStates

Class finesse.restservices.Dialog.ParticipantStates

The list of participant states for voice dialogs.

Field Details

Name	Description
ACCEPTED	The participant has accepted the dialog. This state is applicable to OUTBOUND_PREVIEW dialogs.
ACTIVE	The participant is active on the dialog.
ALERTING	An incoming dialog is ringing on the device.
DROPPED	The participant has dropped from the dialog.
FAILED	The dialog failed (BUSY).
HELD	The participant has held the connection to the dialog.
INITIATED	The phone is dialing at the device.
INITIATING	An outgoing dialog, not yet active, exists on the device.
WRAP_UP	The participant is not in an active state on the dialog. The participant has dropped from the dialog and is wrapping up.

Dialog.ReasonStates

Class finesse.restservices.Dialog.ReasonStates

The list of reason codes for a FAILED dialog participant state. This code can be found in the stateCause field when the participant's state is FAILED. In all other cases, the stateCause is empty.

Field Details

Name	Description
BAD_DESTINATION	The dialog reached a bad destination. For example, making a call to a non-existence extension.
BUSY	The dialog is busy. The dialog failed due to a reason other than the ones listed in this table.
DEVICE_RESOURCE_NOT_AVAILABLE	The device resource for the dialog was not available.
OTHER	All the other reasons. The dialog failed due to a reason other than the ones listed in this table.

Dialog.States

Class finesse.restservices.Dialog.States

The list of states for a dialog.

Field Details

Name	Description
ACCEPTED	The user has accepted an OUTBOUND_PREVIEW dialog.
ACTIVE	The dialog has at least one active participant.
ALERTING	The call is ringing at a device.
DROPPED	The dialog has no active participants.
FAILED	The dialog has failed.
INITIATED	The phone is dialing at the device.
INITIATING	The phone is off the hook at a device.

DialogBase

Class finesse.restservices.DialogBase

Extends finesse.restservices. RestBase Common Parameters

DialogBase is extended into individual JavaScript objects (such as Dialog and MediaDialog) and cannot be used directly. DialogBase defines common utilities that are used by both Dialog and MediaDialog objects.

A dialog is a connection between multiple participants. For example, a regular phone call, a chat, or an email.

Example

```
var _dialogs = _user.getDialogs({
        includeNonVoice: true
    }
    _dialogs.addHandler('load', function() {
        dialog
    })._dialogs.getCollection();
    for (dialog in dialogCollection) {
        if (dialogCollection.hasOwnProperty(dialog)) {
            var _dialog = dialogCollection[dialog];
            _dialog.addHandler(‘change’, function() {
                // TODO: on change of dialog do some thing
            }));
    }
}
```

Methods

getCallType()

Retrieves the call type.

Note This method is deprecated. Use getMediaProperties().callType.

Returns

`{String}` The call type.

updateCallVariables(callvariablesList, options)

Updates the dialog's call variables. This function does not validate the call variables. The client must take care of validating the call variables before using this function.

Parameters

Name	Type	Description	Required
callvariablesList	Object	The call variables are from 1 to 10. For example, callVariable1: value1, callVariable2: value2, and so on.	Yes
options	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Example

```
var callVariablePayload = {
    "callVariable1": "value1",
    "callVariable2": "value2"
};
dialog.updateCallVariables(callVariablePayLoad, {
    error: function() {
        console.log("Error on updating call variable");
    }
});
```

```
var callVariablePayload = {
    "callVariable1": "value1",
    "callVariable2": "value2",
    "user.eccVariable1": "value3"
};
dialog.updateCallVariables(callVariablePayLoad, {
    error: function() {
        console.log("Error on updating call variable");
    }
});
```

getMediaProperties()

Retrieves a list of media properties from the dialog object.

Returns

`{Object}` The call variables and the names are mapped to values. Variables include the following:

- dialedNumber—The number dialed.
- callType—The type of call. Call types include:
    - ACD_IN
    - PREROUTE_ACD_IN
    - PREROUTE_DIRECT_AGENT
    - TRANSFER

    - OTHER_IN
    - OUT
    - AGENT_INSIDE
    - CONSULT
    - CONFERENCE
    - SUPERVISOR_MONITOR
    - OUTBOUND
    - OUTBOUND_PREVIEW

- DNIS—The DNIS provided. For routed calls, this is the route point.
- wrapUpReason—Description of the call.
- queueNumber—The Id of the agent Skill Group queue.
- queueName—Name of the agent Skill Group the that the call is attributed to.
- callKeyCallId—Unique number of the call routed on a particular day.
- callKeyPrefix—The day when the call is routed.
- callKeySequenceNum—The sequence number of the call.
- Variables by name—The name indicates whether it is a call variable or an ECC variable. Call variable names to start with callVariable#, where # refers to 1–10. ECC variable names (both scalar and array) are prepended with the user. ECC variable arrays include an index that is enclosed within square brackets that are located at the end of the ECC array name.
- The following call variables provide additional details about an Outbound Option call:
    - BACampaign
    - BAAccountNumber
    - BAResponse
    - BAStatus
        - PREDICTIVE_OUTBOUND—A predictive outbound call.
        - PROGRESSIVE_OUTBOUND—A progressive outbound call.
        - PREVIEW_OUTBOUND_RESERVATION—The agent is reserved for a preview outbound call.
        - PREVIEW_OUTBOUND—The agent is on a preview outbound call.
    - BADialedListID
    - BATimeZone
    - BABuddyName

getMediaType()

Retrieves the media type.

Returns

`{String}` The media type.

getParticipants()

Retrieves a list of participants from the dialog object.

`{Array}` The array list of participants. The participant entity properties are as follows:

- state—The last participant state in a dialog.
- stateCause—The state cause of the participant.
- mediaAddress—The media address of the participant.
- startTime—The start time of the participant.
- stateChangeTime—The time when the participant state has changed.
- actions—These are the actions that a participant can perform.

getState()

Retrieves the dialog state.

Returns

`{String}` The dialog state.

For additional parameters and methods, see RestBase Common Parameters, on page 496.

DialogLogoutActions

Class finesse.restservices.DialogLogoutActions

Represents the logout actions for media tasks (non-voice dialogs). The actions are performed on the dialog when the user logs out.

Field Details

Name	Description
CLOSE	The user sets the action to close active dialogs when logged out.
TRANSFER	The user sets the action to transfer active dialogs when logged out.

Method

isValidAction(action)

Determines whether the logout action is valid.

Parameters

Name	Type	Description	Required
action	String	The action to evaluate.	Yes

Returns

`{Boolean}` True if the action is valid and false if it is not valid.

Dialogs

Class finesse.restservices.Dialogs

Extends finesse.restservices. RestCollectionBase

Represents a collection of dialog objects.

Example

```
_dialogs = _user.getDialogs({
    onCollectionAdd: _handleDialogAdd,
    onCollectionDelete: _handleDialogDelete,
    onLoad: _handleDialogsLoaded
});

_dialogCollection = _dialogs.getCollection();
for (var dialogId in _dialogCollection) {
    if (_dialogCollection.hasOwnProperty(dialogId)) {
        _dialog = _dialogCollection[dialogId];
        etc...
    }
}
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

Field Details

Name	Description
REQUESTID_REAPER_TIMEOUT	Unique identifier of the request reaper timeout.

Method

getDialogCount(excludeSilentMonitor)

Retrieves the number of dialogs in the collection. The excludeSilentMonitor parameter is provided as an option to exclude dialogs with the call type of SUPERVISOR_MONITOR from the count.

Parameters

Name	Type	Description	Required
excludeSilentMonitor	Boolean	Determines whether the dialogs with the call type of SUPERVISOR_MONITOR is to be excluded from the count.	Yes

Returns

`{Number}` The number of dialogs in the collection.

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

Queue

Class finesse.restservices.Queue

Extends finesse.restservices. RestBase Common Parameters

Represents a queue (or skill group in Unified CCE) and contains the URI, name, and statistics for that queue. Queue statistics include the number of calls in queue, the start time of the longest call in the queue, and the number of agents in each state.

Note This class is only applicable to Unified CCE.

Methods

getId()

Retrieves the queue Id.

Returns

`{String}` The Id of the Queue.

getName()

Retrieves the queue name.

Returns

`{String}` The name of the queue.

getStatistics()

Retrieves the queue statistics. The following are the supported statistics:

- agentsBusyOther
- agentsLoggedOn
- agentsNotReady
- agentsReady
- agentsTalkingInbound
- agentsTalkingInternal
- agentsTalkingOutbound
- agentsWrapUpNotReady

- agentsWrapUpReady
- callsInQueue
- startTimeOfLongestCallInQueue

Individual statistics can be accessed through dot notation. For example, `getStatistics().callsInQueue`.

Returns

`{Object}` The object with different statistics as properties.

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Queues

Class finesse.restservices.Queues

Extends finesse.restservices. RestCollectionBase

Represents a collection of Queue objects. For more information, see Queue, on page 153.

Note External subscriptions to the Queues for queue stats notifications will have a performance impact. This is not qualified for third-party gadget subscriptions.

Example

```
 _queues = _user.getQueues({
    onCollectionAdd: _handleQueueAdd,
    onCollectionDelete: _handleQueueDelete,
    onLoad: _handleQueuesLoaded
});

_queueCollection = _queues.getCollection();
for (var queueId in _queueCollection) {
    if (_queueCollection.hasOwnProperty(queueId)) {
        _queue = _queueCollection[queueId];
        etc...
    }
}
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

Team

Class finesse.restservices.Team

Extends finesse.restservices. RestBase Common Parameters

Represents a team and contains the URI, team name, and the users associated with the team. The Team object does not contain a full user object for each of the team's users, but a summary object that contains the User URI, loginId, firstName, lastName, ReasonCode, and extension parameters.

Example

```
var _team = new finesse.restservices.Team({
    id: id,
    onLoad: _onTeamLoad,
    onChange: _onTeamChange,
    onError: _onTeamError
})
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Methods

getId()

Retrieves the team Id.

Returns

`{String}` The Id of the team.

getName()

Retrieves the team name.

Returns

`{String}` The name of the team.

getUsers(options)

Retrieves a collection of the users in the team.

Parameters

Name	Type	Description
options	Object	Options for the Users collection object. For additional parameters and methods, see RestCollectionBase Common Parame

Returns

`{Object}` The collection of User object representing the users in the team.

For additional parameters and methods, see RestBase Common Parameters, on page 496.

TeamNotReadyReasonCode

Class finesse.restservices.TeamNotReadyReasonCode

Represents a Not Ready reason code that is configured for the team.

Example

```
new_team.getTeamNotReadyReasonCode({
    id: id,
    onLoad: _onTeamLoad,
    onChange: _onTeamChange,
    onError: _onTeamError
})
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Methods

getCategory()

Retrieves the category of the reason code.

Returns

`{String}` The category of the reason code.

getCode()

Retrieves the code associated with the reason code.

Returns

`{String}` The code for the reason code.

getForAll()

Retrieves the forAll property value.

Returns

`{String}` The forAll property value.

getLabel()

Retrieves the label associated with the reason code.

Returns

`{String}` The label associated with the reason code.

getSystemCode()

Retrieves the system code value. The system code is the reserved status of the reason code. This is supported from Cisco Finesse, Release 11.6(1) ES1 onwards.

Returns

`{String}` The value for systemCode.

getUri()

Retrieves the URI value.

Returns

`{String}` The URI value.

TeamNotReadyReasonCodes

Class finesse.restservices.TeamNotReadyReasonCodes

Extends finesse.restservices. RestCollectionBase

Represents the list of Not Ready reason codes configured for the team as a collection of TeamNotReadyReasonCode objects.

Example

```
new_team.getTeamNotReadyReasonCodes({
    id: id,
    onLoad: _onTeamLoad,
    onChange: _onTeamChange,
    onError: _onTeamError
})
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

TeamSignOutReasonCodes

Class finesse.restservices.TeamSignOutReasonCodes

Represents a collection of TeamSignOutReasonCode objects and also exposes methods to operate on the object against the server.

Example

```
new_team.getTeamSignOutReasonCodes({
id: id,
onLoad: _onTeamLoad,
onChange: _onTeamChange,
onError: _onTeamError
})
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

Methods

get()

Retrieves the sign out reason codes.

Returns

`{finesse.restservices.TeamSignOutReasonCodes}` The TeamSignOutReasonCodes object.

getRestUrl()

Retrieves the URL for the SignOutReasonCodes resource.

update(newValues, handlers)

Updates the sign out reason codes with new values.

Parameters

Name	Type	Description	Required
newValues	Object	The new values to be set which is triggered after user signs out such as, reasoncode name, label or code, global status.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.TeamSignOutReasonCodes}` The TeamSignOutReasonCodes object.

Media

Class finesse.restservices.Media

Extends finesse.restservices.RestCollectionBase

Represents a non-voice media channel such as, chat or email. This object can be used to perform various actions in the media channel such as, logging into, or logging out of the media channel, setting the state of the user for the media channel (Ready, Not Ready), setting the maximum allowed number of dialogs for this media, and so on.

Example

```
_mediaCollection = mediaList.getCollection();
if (_mediaCollection.hasOwnProperty(_emailMediaId)) {
    media = _mediaCollection[_emailMediaId];
    media.login({
        maxDialogLimit: 5,
        interruptAction: 'ACCEPT',
        dialogLogoutAction: 'CLOSE',
        handlers: _handlers
    });
    ...
}
}
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

Methods

getDialogLogoutAction()

Retrieves the action that is taken when the agent logs out with dialogs that are associated with the media.

Returns

`{String}` The action taken when dialog logs out. The actions are:

- CLOSE—The dialog is closed.
- TRANSFER—The dialog is transferred to another agent.

getId()

Retrieves the Id.

Returns

`{String}` The Id.

getInterruptAction()

Retrieves the action that is taken when the media is interrupted.

Returns

`{String}` The action taken when media is interrupted. The actions are:

- ACCEPT—The interrupt is accepted and the agent is not allowed to work on tasks in the media.
- IGNORE—The interrupt is ignored and the agent is allowed to work on tasks in the media.

getInterruptible()

Retrieves whether the media is interruptible.

Returns

`{Boolean}` True if interruptible and false if it is not interruptible.

getMaxDialogLimit()

Retrieves the maximum number of dialogs that are allowed in the media.

Returns

`{String}` The maximum number of dialogs in the media.

getMediaDialogs(handlers)

Retrieves the MediaDialogs collection object that is associated with the user in the media.

Example

```
 First call:
    _mediaDialogs = _media.getMediaDialogs({
        onLoad: _handleMediaDialogsLoad,
        onChange: _handleTeamChange,
        onAdd: _handleMediaDialogAdd,
        onDelete: _handleMediaDialogDelete,
        onError: _errorHandler
    });
Subsequent calls on the same object, after the media dialogs are loaded:
    ...
    _mediaDialogsNew = _media.getMediaDialogs();
_dialogsCollection = _mediaDialogsNew.getCollection();
...
```

Parameters

Name	Type	Description	Required
handlers	Object	An object containing callback functions which are invoked when the callback scenario is triggered. To find the list of callback scenarios, see RestCollectionBase Common Parameters, on page 498.	Optional

Returns

`{finesse.restservices.MediaDialogs}` The MediaDialogs object.

getMediaId()

Retrieves the media Id

Returns

`{String}` The media Id.

getName()

Retrieves the media name.

Returns

`{String}` The media name.

getReasonCodeId()

Retrieves the reason code Id.

Returns

`{String}` The reason code Id.

getReasonCodeLabel()

Retrieves the reason code label.

Returns

`{String}` The reason code label.

getRoutable()

Retrieves whether the user is routable in the media.

Returns

`{Boolean}` True if routable, and false if the value is not routable.

getState()

Retrieves the state of the user in the media.

Returns

`{String}` The current (or last fetched) state of the user in the media.

isInterruptible()

Retrieves whether the user is interruptible in the media.

Returns

`{Boolean}` True if the user is interruptible, and false if the user is not interruptible.

isInWorkState()

Retrieves whether the user is in work state in the media.

Returns

`{boolean}` True if the media is in work state, and false if it is not in the work state.

isLoggedIn()

Retrieves whether the user is in any state except LOGOUT in the media.

Returns

`{boolean}` True if the agent is in any state except LOGOUT in the media.

isRoutable()

Retrieves if the user is routable in the media.

Returns

`{Boolean}` True if the user is routable, and false if the user is not routable.

login(params)

Logs the agent into the media.

Parameters

Name	Type	Description	Required
params	Object	An media object with options.	Yes
-->maxDialogLimit	String	The maximum number of dialogs that are allowed in the media. For more information, see getMaxDialogLimit(), on page 542.	Yes
-->interruptAction	String	The action that is taken when the media is interrupted. The actions are: • ACCEPT • IGNORE For more information, see getInterruptAction(), on page 541.	Yes
-->dialogLogoutAction	String	The action taken with the dialogs that are associated with the media when the agent logs out. The actions are: • CLOSE • TRANSFER For more information, see getDialogLogoutAction(), on page 541.	Yes
-->handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Note If maxDialogLimit, interruptAction, and dialogLogoutAction are not present, then the loginOptions specified in the call to finesse.restservices.MediaList.getMedia is used.

Returns

`{finesse.restservices.Media}` The media object.

logout(reasonCode, params)

Logs out the agent from the media.

Parameters

Name	Type	Description	Required
reasonCode	String	The reasonCode for the user to log out of the media. Pass null as the first parameter if no reason code is associated with the log out event.	Yes
params	Object	An media object.	Yes
-->handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.Media}` The media object.

refresh(retries)

Refreshes the media object and optionally refreshes the list of media dialogs that are associated with the object.

Parameters

Name	Type	Description	Required
retries	Integer	The number of attempts to retry synchronizing the media object.	Yes

refreshMediaDialogs()

Refreshes the dialog collection that is associated with the media. The refresh happens only if the media dialogs are initialized.

setMediaOptions(mediaOptions)

Sets the maxDialogLimit, interruptAction, and dialogLogoutAction settings that the application uses for the media. During a failure, these options are set in the new Cisco Finesse server.

Parameters

Name	Type	Description	Required
mediaOptions	Object	An media object with options.	Yes
-->maxDialogLimit	String	The maximum number of dialogs that are allowed in the media. For more information, see #unique_430 unique_430_Connect_42_d15e137.	Yes
-->interruptAction	String	The action that is taken when the media is interrupted. The actions are: • ACCEPT • IGNORE For more information, see #unique_430 unique_430_Connect_42_d15e89.	Yes
-->dialogLogoutAction	String	The action taken with the dialogs that are associated with the media when the agent logs out. The actions are: • CLOSE • TRANSFER For more information, see #unique_430 unique_430_Connect_42_d15e43.	Yes

setRoutable(options)

Sets the routable status in the media.

Parameters

Name	Type	Description	Required
options	Object	An media object with options.	Yes
-->routable	String	Determines whether the agent is routable or not.	Yes
-->handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.Media}` The media object.

setState(newState, reasonCode, params)

Sets the state of the user in the media.

Example

```
_media.setState(finesse.restservices.Media.States.NOT_READY, {
    id: _reasonCodeId
}, {
    handlers: {
        success: _handleStateChangeSuccess,
        error: _handleStateChangeError
    }
});
```

Parameters

Name	Type	Description	Required
newState	String	The new state of the user in the media.	Yes
reasonCode	String	The reasonCode for the user to log out of the media. Pass null as the first parameter if no reason code is associated with the log out event.	Yes
params	Object	An media object with options.	Yes
-->maxDialogLimit	String	The maximum number of dialogs that are allowed in the media. For more information, see #unique_430 unique_430_Connect_42_d15e137.	Yes
-->interruptAction	String	The action that is taken when the media is interrupted. The actions are: • ACCEPT • IGNORE For more information, see #unique_430 unique_430_Connect_42_d15e89.	Yes
-->dialogLogoutAction	String	The action taken with the dialogs that are associated with the media when the agent logs out. The actions are: • CLOSE • TRANSFER For more information, see #unique_430 unique_430_Connect_42_d15e43.	Yes
-->handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.Media}` The media object.

Media.States

Class finesse.restservices.Media.States

Contains the various states that a non-voice media can be set to. This can be used for setting state in finesse.restservices.Media.setState, or comparing the state which is got from finesse.restservices.Media.getState.

Field Details

Field	Description
INTERRUPTED	The dialog has been interrupted by a dialog from another Media Routing Domain (MRD).
LOGIN	The logged-in user on a non-voice media. **Note** This is an action and not a state. A logged-in user is always in a specific state (READY or NOT_READY).
LOGOUT	The user is logged out from the media.
NOT_READY	The user is not ready in the media.
READY	The user is ready for activity in the media.
RESERVED	The user has a dialog coming in, but has not accepted it.
WORK	The user enters this state when failing over from one Cisco Finesse to the other or when failing over from one PG side to the other.

MediaDialog

Class finesse.restservices.MediaDialog

Extends finesse.restservices.DialogBase

Represents a non-voice media dialog and also exposes the methods to perform actions on the media dialog. For example, accepting, starting, pausing, resuming, transferring, closing, and so on.

Example

```
_MediaDialogs = _media.getMediaDialogs({
    onCollectionAdd: _handleDialogAdd,
    onCollectionDelete: _handleDialogDelete,
    onLoad: _handleMediaDialogsLoaded
});
_dialogCollection = _MediaDialogs.getCollection();
for (var mediadialogId in _dialogCollection) {
    if (_dialogCollection.hasOwnProperty(dialogId)) {
        _mediadialog = _dialogCollection[mediadialogId];
        etc...
    }
}
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Methods

setTaskState(action, handlers, target)

Sets the state on a media dialog based on the action given.

Example

```
_mediaDialog.setTaskState(finesse.restservices.MediaDialog.TaskActions.ACCEPT, {
        success: _handleAcceptSuccess,
        error: _handleAcceptError
    },
    null)
```

Parameters

Name	Type	Description	Required
action	String	The action is invoked when the media dialog is set.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes
target	String	The script selector or dialed number to which the dialog is being transferred. Pass the target as null if the task is not related to transfer.	Yes

transfer(target, handlers)

Transfers a media dialog to the specified target.

Name	Type	Description	Required
target	String	The script selector or dialed number to which the dialog is being transferred.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

MediaDialog.States

Class finesse.restservices.MediaDialog.States

Contains the various states in a non-voice media dialog. This can be used to compare the state of the non-voice media dialog, got from the method finesse.restservices.MediaDialog.getState.

Field Details

Field	Description
ACCEPTED	The user accepted the offered dialog.

Field	Description
ACTIVE	The dialog has at least one active participant and the user has started working on the accepted dialog.
CLOSED	The dialog is ended.
INTERRUPTED	The dialog is interrupted by a dialog from another MRD.
OFFERED	The dialog is offered to a user.
PAUSED	The user has paused an active dialog.
UNKNOWN	The Cisco Finesse server or PG has recovered a dialog after a failure. It does not have enough information to build a complete set of actions for the task; so it only allows the user to end the task.
WRAPPING_UP	The user is wrapping up for a dialog.

MediaDialog.TaskActions

Class finesse.restservices.MediaDialog.TaskActions

Contains the various task actions that can be performed on a non-voice media dialog. This is used for setting task state in finesse.restservices.MediaDialog.setTaskState.

Field Details

Field	Description
ACCEPT	The user accepts an incoming task.
CLOSE	The user ends a task.
PAUSE	The user pauses work on an active task.
RESUME	The user resumes work on a paused task.
START	The user starts work on an accepted task.
TRANSFER	The user transfers an accepted, active, or paused task to another script selector or dialed number (target).
WRAP_UP	The user wraps up work for a task.

MediaDialogs

Class finesse.restservices.MediaDialogs

Extends finesse.restservices.Dialogs

Represents the list of non-voice media dialogs in a media channel. Fetches the list from the corresponding media, finesse.restservices.Media.getMediaDialogs. Extracts the Individual media dialogs from the media list using their Id.

Example

```
_MediaDialogs = _media.getMediaDialogs({
    onCollectionAdd: _handleDialogAdd,
    onCollectionDelete: _handleDialogDelete,
    onLoad: _handleMediaDialogsLoaded
});

_dialogCollection = _MediaDialogs.getCollection();
for (var dialogId in _dialogCollection) {
    if (_dialogCollection.hasOwnProperty(dialogId)) {
        _dialog = _dialogCollection[dialogId];
        etc...
    }
}
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

Method

getMedia()

Retrieves the associated media object.

Returns

`{finesse.restservices.Media}` The Media object.

MediaList

Class finesse.restservices.MediaList

Extends finesse.restservices.RestCollectionBase

Represents the list of non-voice media such as chat and email for an agent. The media list available for an agent can be procured from the finesse.restservices.User object, which corresponds to the agent. Individual media channels can be extracted from the media list using their Id.

Example

```
mediaList = _user.getMediaList({
    onCollectionAdd: _handleMediaAdd,
    onCollectionDelete: _handleMediaDelete,
    onLoad: _handleMediaListLoaded
});

_mediaCollection = mediaList.getCollection();
for (var mediaId in _mediaCollection) {
    if (_mediaCollection.hasOwnProperty(mediaId)) {
        media = _mediaCollection[mediaId];
        etc...
    }
}
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

Method

getMedia(options)

Retrieves a specific media with the Id passed from the MediaList collection.

Example

```
var media = mediaList.getMedia({
    id: mediaId,
    onLoad: _onloadCallback,
    onChange: _onChangeCallback,
    onAdd: _onAddCallback,
    onDelete: _onDeleteCallback,
    onError: _onErrorCallback,
    mediaOptions: {
        maxDialogLimit: _dialogLimit,
    }
});
```

Parameters

Name	Type	Description
options	Object	Options for the media object. For additional parameters and methods, see RestBase Common Para

Returns

`{finesse.restservices.Media}` The Media object.

MediaOptionsHelper

Class finesse.restservices.MediaOptionsHelper

Synchronizes media login options after recovering from a connection or system failure. This class ensures that when the Cisco Finesse server application fails over, it has the same maxDialogLimit, interruptAction, and dialogLogoutAction as the previous Cisco Finesse server.

Example

```
_optionsHelper = new MediaOptionsHelper(_media, _mediaOptions);
```

Method

init(media, mediaOptions)

Initializes a helper class that is used to recover media objects following a connectivity failure or component failure. These failures may be related to either Cisco Finesse or Cisco Unified CCE services or both. Initializes the failover helper to manage the recovery of the given media object.

Parameters

Name	Type	Description	Required
media	Object	The media object to recover.	Yes
mediaOptions	Object	The options for the media object.	Yes

Name	Type	Description	Required
-->maxDialogLimit	String	The maximum number of dialogs that are allowed in the media. For more information, see #unique_437 unique_437_Connect_42_d15e137.	Yes
-->interruptAction	String	The action that is taken when the media is interrupted. The actions are: • ACCEPT • IGNORE For more information, see #unique_437 unique_437_Connect_42_d15e89.	Yes
-->dialogLogoutAction	String	The action taken with the dialogs that are associated with the media when the agent logs out. The actions are: • CLOSE • TRANSFER For more information, see #unique_437 unique_437_Connect_42_d15e43.	Yes

MediaOptionsHelper.States

Class finesse.restservices.MediaOptionsHelper.States

Contains the various MediaOptionsHelper state values.

Field Details

Name	Description
MONITORING_OPTIONS	The media is synchronized with the Cisco Finesse server and the options are monitored for changes.
SETTING_OPTIONS	The media login request is sent to Cisco Finesse to set the correct media options.

MediaPropertiesLayout

Class finesse.restservices.MediaPropertiesLayout

Extends finesse.restservices. RestBase Common Parameters

Represents the appearance of media properties in the call control gadget on the agent or supervisor desktop. Media properties are carried in Dialog objects. Administrators can create and customize multiple layouts for media properties.

Example

```
_mediaPropLayout = new finesse.restservices.MediaPropertiesLayouts({
"onLoad": _onLoadHandler,
"onError": _onErrorHandler,
"onChange": _onChangeHandler
});
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Methods

get()

Retrieves the media properties layout. This call re-queries the server and refreshes the layout object.

Returns

`{finesse.restservices.MediaPropertiesLayout}` The MediaPropertiesLayout object.

getData()

Retrieves the data for the object. Performs safe conversion from raw API data to ensure the returned layout. The object has a header with correct entry fields and exactly two columns with lists of entries.

Returns

`{finesse.restservices.MediaPropertiesLayout.Object}` The data in columns (unless only one defined).

getDescription()

Retrieves the description.

Returns

`{String}` The description.

getName()

Retrieves the name.

Returns

`{String}` The name.

getType()

Retrieves the layout type DEFAULT or CUSTOM.

Returns

`{String}` The layout type DEFAULT or CUSTOM.

Script Selectors

Class finesse.restservices.ScriptSelector

Class finesse.restservices.RestBase. For more information, see REST Base, on page 502.

Represents a script selector for a specific MRD type. The MRD type can be either `voice` or `digitalChannels`.

Methods

getMRDID()

Returns a unique media routing domain (MRD) ID.

getName()

Returns the name of the media channel configured in Unified CCE.

getDialedNumber()

Returns the dialed number also called as script selector, that is submitted with Task Routing task request through Customer Collaboration Platform.

getDescription()

Returns the additional information about the dialed number.

Class finesse.restservices.ScriptSelectors

Extends finesse.restservices.RestCollectionBase. For more information, see RestCollectionBase Common Parameters, on page 498.

Represents the collection of script selectors for a specific MRD type or all the MRD types. The MRD types can be `voice`, `digitalChannels`, and `all`. When no parameter is specified, `all` option is considered.

Example:

```
var _scriptSelectorsVoice = new finesse.restservices.ScriptSelectors({ 'mrdType' :
'voice'}).getScriptSelectors();
var _scriptSelectorsDigitalChannels = new finesse.restservices.ScriptSelectors({ 'mrdType'
 : 'digitalchannels'}).getScriptSelectors();
var _scriptSelectorsAll = new finesse.restservices.ScriptSelectors().getScriptSelectors();
var _scriptSelectorsAll2 = new finesse.restservices.ScriptSelectors({ 'mrdType' :
'all'}).getScriptSelectors();

for(var i = 0; i < _scriptSelectorsVoice.length; ++i) {
    var _mrdID = _scriptSelectorsVoice[i].getMRDID();
    var _name = _scriptSelectorsVoice[i].getName();
    var _dialedNumber = _scriptSelectorsVoice[i].getDialedNumber();
    var _description = _scriptSelectorsVoice[i].getDescription();
    ...
 }
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

ChatConfig

Class finesse.restservices.ChatConfig

Represents the ChatConfig object and exposes methods to operate on the object against the server. The ChatConfig object is a container element that holds the Cisco Finesse chat configuration and URLs of the primary and secondary chat servers.

Example

```
_chatcfg = new finesse.restservices.ChatConfig({
"onLoad": _onLoadHandler,
"onError": _onErrorHandler,
"onChange": _onChangeHandler
});
```

Field Details

Name	Description
supportsSubscriptions	Determines whether the object supports subscriptions. For more information, see Subscription Support, on page 500.

Methods

get()

Retrieves the ChatConfig settings.

Returns

`{finesse.restservices.ChatConfig}` The ChatConfig object.

getPrimaryNode()

Retrieves the primary node of the chat server.

Returns

`{String}` The primary node of the chat server.

getRestClass()

Retrieves the REST class for the ChatConfig object.

getRestType()

Retrieves the REST type for the ChatConfig object.

getRestUrl()

Retrieves the URL for the ChatConfig resource.

Returns

`{String}` The URL for the ChatConfig resource.

getSecondaryNode()

Retrieves the secondary node (if any) of the chat server.

Returns

`{String}` The secondary node of the chat server.

update(chatSettings, handlers)

Updates the chat configuration settings.

Parameters

Name	Type	Description	Required
chatSettings	Object	The chat server settings which you want to configure.	Yes
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Optional

Returns

`{finesse.restservices.ChatConfig}` The ChatConfig object.

ECCVariableConfig

Class finesse.restservices.ECCVariableConfig

Represents the ECCVariableConfig object and also exposes methods to operate on the object against the server.

Example

```
_eccvariableconfig = new finesse.restservices.ECCVariableConfig({
onLoad:  _onLoadECCVariableConfig
})
```

Field Details

Name	Description
supportsSubscriptions	Determines whether the object supports subscriptions. For more information, see Subscription Support, on page 500.

Methods

get()

Retrieves the ECCVariableConfig settings.

Returns

`{finesse.restservices.ECCVariableConfig}` The ECCVariableConfig object.

getRestClass()

Retrieves the REST class for the current object and this is the ECCVariableConfig object.

getRestType()

Retrieves the REST type for the current object and this is ECCVariableConfig.

getRestUrl()

Retrieves the URL for the ECCVariableConfig resource.

Returns

`{String}` The URL for the ECCVariableConfig resource.

Contact

Class finesse.restservices.Contact

Extends finesse.restservices.RestBase Common Parameters

Represents the collection of Contact objects. A Contact is a single entry in a phone book, consisting of a first name, last name, phone number, and description.

Example

```
_contacts = _phonebook.getContacts({
    onCollectionAdd: _handleContactAdd,
    onCollectionDelete: _handleContactDelete,
    onLoad: _handleContactsLoaded
});
_contactCollection = _contacts.getCollection();
for (var contactId in _contactCollection) {
    if (_contactCollection.hasOwnProperty(contactId)) {
        _contact = _contactCollection[contactId];
        etc...
    }
}
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Methods

getDescription()

Retrieves the description for a contact.

Returns

`{String}` The description for a contact.

getFirstName()

Retrieves the first name of a contact.

Returns

`{String}` The first name of a contact.

getLastName()

Retrieves the last name of a contact.

Returns

`{String}` The last name of a contact.

getPhoneNumber()

Retrieves the phone number.

Returns

`{String}` The phone number.

Contacts

Class finesse.restservices.Contacts

Extends finesse.restservices. RestCollectionBase

Represents the Contacts collection object and also exposes methods to operate on the object against the server.

Example

```
 _contacts = _phonebook.getContacts({
    onCollectionAdd: _handleContactAdd,
    onCollectionDelete: _handleContactDelete,
    onLoad: _handleContactsLoaded
});

_contactCollection = _contacts.getCollection();
for (var contactId in _contactCollection) {
    if (_contactCollection.hasOwnProperty(contactId)) {
        _contact = _contactCollection[contactId];
        etc...
    }
}
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

InterruptActions

Class finesse.restservices.InterruptActions

InterruptActions are used during non-voice media login. These consist of the actions to be taken when the dialog has been interrupted by a dialog from another Media Routing Domain (MRD). Dialogs can be interrupted if they are in the ACTIVE, PAUSED, or WRAPPING UP states. While a dialog is interrupted, all actions for that dialog are disabled. The following are the action values.

- ACCEPT
- IGNORE

Field Details

Field	Description
ACCEPT	The interrupt is accepted and the agent is not allowed to work on dialogs.
IGNORE	The interrupt is ignored and the agent is allowed to work on dialogs.

Method

isValidAction(action)

Determines whether the action is valid dialog logout.

Parameters

Name	Type	Description	Required
action	String	Action to evaluate the dialog.	Yes

Returns

`{Boolean}` True if the action is valid; otherwise it returns false.

PhoneBook

Class finesse.restservices.PhoneBook

Extends finesse.restservices. RestBase Common Parameters

PhoneBook is a list of contacts available to a user for a quick dial. PhoneBooks are assigned globally (to all agents) or to specific teams.

Example

```
_phoneBooks = _user.getPhoneBooks({
onCollectionAdd: _handlePhoneBookAdd,
onCollectionDelete: _handlePhoneBookDelete,
onLoad: _handlePhoneBooksLoaded
});
_phoneBookCollection = _phoneBooks.getCollection();
for (var phoneBookId in _phoneBookCollection) {
if (_phoneBookCollection.hasOwnProperty(phoneBookId)) {
_phoneBook = _phoneBookCollection[phoneBookId];
etc...
}
}
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Methods

getContacts(handlers)

Retrieves the collection of contact objects that is part of the PhoneBook.

Parameters

Name	Type	Description	Required
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes

Returns

`{finesse.restservices.Contacts}` The Contacts object.

getEmbeddedContacts()

Retrieves either a URI or collection of contacts.

Returns

`{String}` The URI or finesse.restservices.Contacts object.

getName()

Retrieves the name of the PhoneBook.

Returns

`{String}` The name of the PhoneBook.

getType()

Retrieves the PhoneBook type. The following are the two types of PhoneBooks.

- GLOBAL—By default, available to all agents. Cisco Finesse supports a maximum of 10 global PhoneBooks.
- TEAM—The administrator assigns the PhoneBooks to a specific team. Cisco Finesse supports a maximum of 300 team PhoneBooks.

Returns

`{String}` The PhoneBook type.

PhoneBooks

Class finesse.restservices.PhoneBooks

Extends finesse.restservices.RestCollectionBase

Represents the phone book collection object.

Example

```
_phoneBooks = _user.getPhoneBooks({
    onCollectionAdd: _handlePhoneBookAdd,
    onCollectionDelete: _handlePhoneBookDelete,
    onLoad: _handlePhoneBooksLoaded
});

_phoneBookCollection = _phoneBooks.getCollection();
for (var phoneBookId in _phoneBookCollection) {
    if (_phoneBookCollection.hasOwnProperty(phoneBookId)) {
        _phoneBook = _phoneBookCollection[phoneBookId];
        etc...
    }
}
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

ReasonCodeLookup

Class finesse.restservices.ReasonCodeLookup

Extends finesse.restservices. RestBase Common Parameters

Retrieves the ReasonCode using the reason code value and category provided by the user.

Method

lookupReasonCode(handlers, reasonCodeValue, reasonCodeCategory)

Performs a GET operation against the Cisco Finesse server, for looking up the reason code with its reason code value, and category.

Example

```
new finesse.restservices.ReasonCodeLookup().lookupReasonCode({
    success: _handleReasonCodeGet,
    error: _handleReasonCodeGetError
}, '32762', 'NOT_READY');
_handleReasonCodeGet(_reasonCode) {
    var id = _reasonCode.id;
    var uri = _reasonCode.uri;
    var label = _reasonCode.label;
    ...
}
```

Parameters

Name	Type	Description	Required
handlers	Object	An object containing the handlers for the request. For more information on handlers, see Request Handlers, on page 493.	Yes
reasonCodeValue	String	The lookup code for the reasonCode value.	Yes
reasonCodeCategory	String	The lookup category for the reasonCode. The possible values are NOT_READY and LOGOUT.	Yes

ReasonCodes

Class finesse.restservices.ReasonCodes

Represents an array of reasonCode objects. An object of this class is passed as a parameter for a success callback after fetching the reasonCodes available for the user using finesse.restservices.User.getNotReadyReasonCodes or finesse.restservices.User.getSignOutReasonCodes.

Example

```
_user.getNotReadyReasonCodes({
    success: handleNotReadyReasonCodesSuccess,
    error: handleNotReadyReasonCodesError
});

handleNotReadyReasonCodesSuccess = function(reasonCodes) {
    for (var i = 0; i < reasonCodes.length; i++) {
        var reasonCode = reasonCodes[i];
        var reasonCodeId = reasonCode.id;
        var reasonCodeLabel = reasonCode.label;
    }
}
}
```

For more information on handlers, see Request Handlers, on page 493.

SystemInfo

Class finesse.restservices.SystemInfo

Extends finesse.restservices. RestBase Common Parameters

Represents the SystemInfo object. For more information, see SystemInfo, on page 563.

Example

```
var systeminfo = new finesse.restservices.SystemInfo('',{

    onLoad: handleSystemInfoOnLoad,
    onChange: handleSystemInfoOnChange
});

systeminfo.getCtiMMode();
```

Methods

getAlternateHost()

Retrieves the FQDN of the alternate Finesse host and this is used for failover purposes.

Returns

`{String}` The FQDN (if properly configured) of the alternate node, defaults to primary if no match is found, and undefined for the single-node deployments. For example, if the desktop is connected to the primary Finesse node, this method will return FQDN of the secondary Finesse node, and vice-versa.

getCtiMMode()

Retrieves the status of the CTI server in maintenance mode.

Note This is only supported for Unified CCE deployments with CTI protocol version 24 and above.

Example

```
systeminfo.getCtiMMode();
```

Returns

`{Boolean}` True if the CTI server is in maintenance mode, else false.

getCtiTimeInMMode()

Retrieves the total time (in seconds) that the CTI server has been in maintenance mode.

The time will be negative if the CTI server to which Finesse is connected is not in maintenance mode.

Note This is only supported for Unified CCE deployments with CTI protocol version 24 and above.

Example

```
systeminfo.getCtiTimeInMMode();
```

Returns

`{Number}` The total time (in seconds) that the CTI server is in maintenance mode.

getCtiServers()

Retrieves the list of CTI servers that Cisco Finesse is connected to.

Example

```
systeminfo.getCtiServers();
```

Returns

`{Obejct}` The list of CTI servers that Cisco Finesse is connected to with the following details:

- hostname—The hostname of the CTI server.
- connectedDuration—The total time (in seconds) that the Cisco Finesse is connected to this particular CTI server.
- active—True if the Cisco Finesse is connected to the active CTI server, else false.

Note

- Indicates whether the Cisco Finesse is connected to the active or stand-by CTI server, for Unified CCE deployments with CTI protocol version 24 and above.
- Indicates the CTI server on which the Cisco Finesse is connected, for Unified CCX and Unified CCE deployments with CTI protocol version prior to 24.

getCtiVersion()

Retrieves the CTI version for the current deployment.

Example

```
systeminfo.getCtiVersion();
```

Returns

`{String}` The CTI version used in the Cisco Finesse to connect the CTI server.

getCurrentTimestamp()

Retrieves the current timestamp from the SystemInfo object. This is used to calculate the time drift between the server and the client.

Example

```
systeminfo.getCurrentTimestamp();
```

Returns

`{String}` The time (GMT) in the format: yyyy-MM-dd'T'HH:mm:ss'Z'

getDeploymentType()

Retrieves the deployment type Unified CCE or Unified CCX.

Example

```
systeminfo.getDeploymentType();
```

Returns

`{String}` The deployment type, which is either Unified CCE or Unified CCX.

getFinesseMMode()

Retrieves the status of the maintenance mode for the particular Cisco Finesse server.

Example

```
systeminfo.getFinesseMMode();
```

Returns

`{Boolean}` True if the Cisco Finesse server is in maintenance mode, else false.

getFinesseTimeInMMode()

Retrieves the total time (in seconds) that the Cisco Finesse server is in maintenance mode.

The time will be negative if the Cisco Finesse server is not in Finesse maintenance mode.

Example

```
systeminfo.getFinesseTimeInMMode();
```

Returns

`{Number}` The total time (in seconds) that the Cisco Finesse server is in maintenance mode.

getHeartbeatInterval()

Retrieves the ctiHeartbeatInterval for the current deployment. This represents the interval of heartbeats between the Cisco Finesse server and CTI server (in seconds) that helps in failover time calculation.

Returns

`{String}` The ctiHeartbeatInterval between the Cisco Finesse server and the CTI server.

getLastCTIHeartbeatStatus()

Retrieves the lastCTIHeartbeatStatus. The lastCTIHeartbeatStatus provides the success or failure of the last heartbeat sent to the CTI server. This can be used to determine whether the Cisco Finesse side is healthy or not. If three consecutive heartbeats fail, the CTI server gets disconnected.

- success—The last CTI heartbeat was successful.

- failure—The last CTI heartbeat was unsuccessful.

Returns

`{finesse.restservices.SystemInfo.lastCTIHeartbeatStatus}` The last Heartbeat status (success or failure) to CTI.

getlicense()

Retrieves the license and is applicable only to Unified CCX deployment.

Returns

`{String}` The Unified CCX deployment license details. Otherwise, an empty string.

getPeripheralId()

Retrieves the peripheral Id that Cisco Finesse is connected to. The peripheral Id refers to the Id of the PG Routing Client (PIM).

Returns

`{String}` The Id of the Unified CCE peripheral to which Cisco Finesse is connected. For Unified CCX, it returns an empty string.

getStatus()

Retrieves the status of the Cisco Finesse server. The possible values are:

- IN_SERVICE
- OUT_OF_SERVICE

Returns

`{finesse.restservices.SystemInfo.Statuses}` The system status.

getStatusReason()

Retrieves the reason for Cisco Finesse being OUT_OF_SERVICE.

Returns

`{String}` Returns the status reason if the Cisco Finesse is OUT_OF_SERVICE. Otherwise, an empty string when Cisco Finesse status is IN_SERVICE.

getSystemAuthMode()

Retrieves the system authentication mode for the current deployment. The possible values are SSO or non-SSO.

Returns

`{String}` The system authentication mode for the current deployment.

getXmppDomain()

Retrieves the XMPP domain of the system. The XMPP servers such as Openfire require to identify the domain that they serve. Hence, configure the XMPP domain which is the JID value.

Returns

`{String}` The XMPP domain.

getXmppPubSubDomain()

Retrieves the XMPP PubSub domain of the system. For third-party applications to identify the domain of the server PubSubDomain is required.

Returns

`{String}` The XMPP PubSub domain.

isSingleNode()

Confirms whether the deployment is a single-node deployment by checking for the existence of the secondary node in the SystemInfo.

Returns

`{Boolean}` True for single-node deployments, else false.

SystemInfo.Statuses

Class finesse.restservices.SystemInfo.Statuses

Represents the system information status values. The possible values are IN_SERVICE and OUT_OF_SERVICE.

Field Details

Field	Description
IN_SERVICE	The system is in service and usual operations are accepted.
OUT_OF_SERVICE	The system is out of service, and usual operations result in a 503 Service Unavailable response.

WrapUpReason

Class finesse.restservices.WrapUpReason

Extends finesse.restservices. RestBase Common Parameters

Represents the reasons that agents can apply to calls. The administrator configures the WrapUp reasons to be available globally to all agents or only to specific teams. Finesse supports WrapUp functionality for incoming calls, outgoing calls, and outbound option dialer calls (Finesse does not support outbound option direct preview mode).

WrapUp reasons are set on a per-call basis. If you apply a WrapUp reason for a call, the same is reflected on the desktops of all other participants (agents) of the call. Finesse desktop users can enter a WrapUp reason during a call or while you are in WrapUp state after the call ends (this includes usual call termination, transfer, and conference drop scenarios).

The WrapUp reason is a code and the description identifies a particular reason that a user is in WORK (WrapUp) mode.

Example

```
_wrapUpReasons = _user.getWrapUpReasons({
    onCollectionAdd: _handleWrapUpReasonAdd,
    onCollectionDelete: _handleWrapUpReasonDelete,
    onLoad: _handleWrapUpReasonsLoaded
});
_wrapUpReasonCollection = _wrapUpReasons.getCollection();
for (var wrapUpReasonId in _wrapUpReasonCollection) {
    if (_wrapUpReasonCollection.hasOwnProperty(wrapUpReasonId)) {
        _wrapUpReason = _wrapUpReasonCollection[wrapUpReasonId];
        etc...
    }
}
```

For additional parameters and methods, see RestBase Common Parameters, on page 496.

Method

getLabel()

Retrieves the WrapUp reason label.

Returns

`{String}` The WrapUp reason label.

WrapUpReasons

Class finesse.restservices.WrapUpReasons

Extends finesse.restservices.RestCollectionBase

Represents the collection of WrapUpReasons objects.

Example

```
_wrapUpReasons = _user.getWrapUpReasons({
    onCollectionAdd: _handleWrapUpReasonAdd,
    onCollectionDelete: _handleWrapUpReasonDelete,
    onLoad: _handleWrapUpReasonsLoaded
});

_wrapUpReasonCollection = _wrapUpReasons.getCollection();
for (var wrapUpReasonId in _wrapUpReasonCollection) {
    if (_wrapUpReasonCollection.hasOwnProperty(wrapUpReasonId)) {
        _wrapUpReason = _wrapUpReasonCollection[wrapUpReasonId];
        etc...
    }
}
```

For additional parameters and methods, see RestCollectionBase Common Parameters, on page 498.

ShortcutKey Service

Allows gadgets or components to create shortcut keys for any component or gadget-related actions.

Example

```
finesse.shortcutkey.ShortcutKeyService.registerShortcutKey(arr);
```

Methods

getShorcutKeys()

Retrieves all the registered shortcut keys.

Example

```
finesse.shortcutkey.ShortcutKeyService.getShortcutKeys();
```

Returns

`{Array}` The array of objects. The objects content includes the following:

- `{String}` accessKey—The key combination for the shortcut.
- `{String}` actionName—The name of the action or operation performed by the assigned shortcut keys.
- `{String}` componentName—The name of the functionality, component, or the gadget.
- `{Boolean}` conflict—Determines whether the shortcut key is conflicting with another shortcut key.
- `{Enum}` executionScope—Determines the execution scope.
- `{Function}` handler—The function that is invoked when the shortcut keys are pressed.
- `{String}` id—Unique identifier of the gadget.
- `{Boolean}` isPageLevel—Determines whether the shortcut key is at the page level.
- `{String}` key—The main key to be combined with modifier keys.
- `{String}` modifierKeys—The modifier key is used commonly in keyboard shortcuts on the host platform.
- `{String}` type—Determines whether the shortcut key runs on component or gadget.

init()

Initiates the ShortcutKeyService for the Container or the gadgets.

Example

```
finesse.shortcutkey.ShortcutKeyService.init();
```

registerShortcutKey(keys)

Registers the shortcut keys for the components or the gadgets. A key combination consists of a main key and a set of modifier keys. The main key is specified by its key character - `key`. A modifier key is `Shift`, `Ctrl`, `Alt`, or the combination.

Parameters

Name	Type	Description	Required
keys	Array	The array of key objects.	Yes

Example

```
finesse.shortcutkey.ShortcutKeyService.init();
finesse.shortcutkey.ShortcutKeyService.registerShortcutKey(
    [{
        "id": "cisco_sample_gadget",
        "componentName": "Sample gadget",
        "actionName": "Sample gadget action name",
       "modifierKeys": finesse.shortcutkey.ShortcutKeyService.CONSTANTS.MODIFIER_KEYS_CTRL,

        "key": "e",
        "executionScope": "activeTab",
        "handler": function() {}
    }]);
```

The following table lists the shortcut key registration payload details.

Name	Type	Description	Required
id	String	Unique identifier of the gadget. Format: companyName_gadgetId_functionId	Optional
componentName	String	The name of the functionality, component, or the gadget.	Yes
actionName	String	The name of the action or operation performed by the assigned shortcut keys.	Yes
modifierKeys	String	The modifier key is used commonly in keyboard shortcuts on the host platform. The keyboard modifier key combinations are: • Ctrl + Shift (default) • Alt + Shift • Ctrl + Alt • Ctrl • Shift • Alt These are predefined in ShortcutKeyService.CONSTANTS. For more information on predefined modifier keys, see ShortcutKeyService.CONSTANTS, on page 571.	Optional
key	String	The main key to be combined with modifier keys. For example, Ctrl + Shift + e where Ctrl and Shift are the modifiers keys, and e is the main key.	Yes

Name	Type	Description	Required
executionScope	Enum	Determines the execution scope. • activeTab: If the gadget is in currently active tab, only then the shortcut keys of that gadget are run. • activeFrame: If the gadget is in focus, only then the shortcut keys of that gadget are run. For more information on CONSTANTS, see ShortcutKeyService.CONSTANTS, on page 571.	Yes
handler	Function	The function that is invoked when the shortcut keys are pressed.	Yes

ShortcutKeyService.CONSTANTS

The following table lists the predefined modifier keys.

Shortcut Key	Modifier Key
Ctrl + Shift	`finesse.shortcutkey.ShortcutKeyService.CONSTANTS.MODIFIER_KEYS.CTRL_SHIFT`
Alt + Shift	`finesse.shortcutkey.ShortcutKeyService.CONSTANTS.MODIFIER_KEYS.ALT_SHIFT`
Ctrl + Alt	`finesse.shortcutkey.ShortcutKeyService.CONSTANTS.MODIFIER_KEYS.CTRL_ALT`
Ctrl	`finesse.shortcutkey.ShortcutKeyService.CONSTANTS.MODIFIER_KEYS.CTRL`
Shift	`finesse.shortcutkey.ShortcutKeyService.CONSTANTS.MODIFIER_KEYS.SHIFT`
Alt	`finesse.shortcutkey.ShortcutKeyService.CONSTANTS.MODIFIER_KEYS.ALT`

Note

- When shortcut keys are used, the callback that is registered by the gadget is run. The gadgets or components perform a specific action on callback.
- Components or gadgets register the shortcut keys on a successful sign in to Finesse desktop.
- After deploying the third-party gadgets, sign in as an agent and as a supervisor to check if there are any shortcut key conflicts. Resolve them if any.

sendKeyupEvent(keyEvent)

Sends the Keyup event object to the Finesse container. If there is any custom iFrame created by the gadget that is not controlled by Finesse, then the Finesse shortcut key framework cannot capture the KeyupEvent from that custom iFrame to run the shortcut keys.

The Keyup event occurs when a keyboard key is released. The Keyup event object is captured inside the child iFrame and propagated to its immediate parent. The parent again has to propagate the event until the event reaches the Finesse container. When the immediate parent is the Finesse container, then use sendKeyupEvent to propagate the event to Finesse container. Param object has to be serializable and cannot contain any functions.

Example

```
var keyEvent = {
    ctrlKey: event.ctrlKey,
    altKey: event.altKey,
    shiftKey: event.shiftKey,
    keyCode: event.keyCode,
    key: event.key
};

sendKeyupEvent(keyEvent);
```

Parameters

Name	Type	Description	Required
keyEvent	Object	The key event sent to the Finesse container.	Yes

Shortcut Keys List

The following table lists out-of-the-box agent-specific shortcut keys which should not be used by third-party applications. If the same shortcut keys are used it results in conflict.

Table 12: Agent Shortcut Keys List (Windows)

Group	Action	Shortcut Key
Agent State	Ready for Call	Ctrl + Alt + R
	Not Ready for Call	Ctrl + Alt + N
	Open Digital Channel State Control	Ctrl + Shift + L
	Ready for All Digital Channels	Ctrl + Shift + V
	Not Ready for All Digital Channels	Ctrl + Shift + Z
Application	Switch between Popover	Ctrl + Alt + P
	Maximize/Restore view	Ctrl + Shift + 0

Group	Action	Shortcut Key
Call Handling	Make New Call	Ctrl + Alt + O
	Direct Transfer Call	Ctrl + Alt + Q
	Open Keypad (DTMF)	Ctrl + Alt + K
	Open Consult	Ctrl + Alt + C
	Wrap-Up Call	Ctrl + Alt + W
	Reclassify Call	Ctrl + Alt + Y
	Schedule Callback	Ctrl + Alt + S
	Answer/Accept Call	Ctrl + Alt + A
	Close - Remove Record from Campaign	Ctrl + Alt + J
	Reject - Return Record to Campaign/Close this Callback	Ctrl + Alt + U
	End Call	Ctrl + Alt + E
	Hold Call	Ctrl + Alt + V
	Retrieve Call	Ctrl + Alt + G
	Transfer Call	Ctrl + Alt + X
	Conference Call	Ctrl + Alt + H
Desktop Chat	Toggle, Minimize and Maximize Chat Window	Ctrl + Shift + 1
	Open Desktop Chat	Ctrl + Shift + 3
Edit Call Variable	Save Edited Call Variable Values	Ctrl + Alt + M
	Revert Edited Call Variable Values	Ctrl + Alt + Z
Keyboard Shortcuts	Keyboard Shortcuts List	Ctrl + Alt + F
Navigation	Home	Ctrl + Alt + 1
	My History	Ctrl + Alt + 2
	My Statistics	Ctrl + Alt + 3
	Manage Customer	Ctrl + Alt + 4
Send Error Report	Send Error Report	Ctrl + Shift + 2
Sign Out	Sign Out	Ctrl + Alt + L
Agent State (for Unified CCE)	Ready for Email	Ctrl + Shift + 4
	Ready for Chat	Ctrl + Shift + 5
	Not Ready for Email	Ctrl + Shift + 6
	Not Ready for Chat	Ctrl + Shift + 7

Table 13: Supervisor Shortcut Keys List (Windows)

Group	Action	Shortcut Key
Call Handling	Barge in Call	Ctrl + Alt + B
	Drop Participant	Ctrl + Alt + D
Team Message	Open Team Message Window	Ctrl + Shift + Y
Team Performance	Select Team	Ctrl + Shift + F

Utilities

Class finesse.utilities.Utilities

Collection of utility methods to deal with various text-related activities.

Methods

b64Decode(input)

Decodes a Base64 encoded string.

Note The output is assumed to be UTF-8, and only the first 8 bits of each output element is significant.

Example

```
finesse.utilities.Utilities.b64Decode(_base64String)
```

Parameters

Name	Type	Description	Required
input	String	The string to decode to Base64.	Yes

Returns

`{String}` The decoded string.

b64Encode(input)

Encodes a string to Base64.

Note The input is assumed to be UTF-8, and only the first 8 bits of each output element is significant.

Example

```
finesse.utilities.Utilities.b64Encode(a + ':' + b)
```

Parameters

Name	Type	Description	Required
input	String	The string to encode to Base64.	Yes

Returns

`{String}` The encoded string.

buildTimeString(timeInMs)

Builds a string that specifies the time in minutes and seconds.

Example

```
finesse.utilities.Utilities.buildTimeString(70000)
```

Parameters

Name	Type	Description	Required
timeInMs	Integer	The time in milliseconds.	Yes

Returns

`{String}` The time in MINUTES:SECONDS. For example, 11:23.

buildTimeStringWithOptionalHours(timeInMs)

Builds a string that specifies the time in minutes, seconds, and optionally hours.

Example

```
finesse.utilities.Utilities.buildTimeStringWithOptionalHours(11170000)
```

Parameters

Name	Type	Description	Required
timeInMs	Integer	The time in milliseconds.	Yes

Returns

`{String}` The time in HOURS:MINUTES:SECONDS or MINUTES:SECONDS. For example, 01:11:23 or 11:23.

buildTotalTimeString(adjustedServerTimeInMs, callStartTimeInMs)

Builds a string that specifies the total call time in minutes and seconds.

Example

```
finesse.utilities.Utilities.buildTotalTimeString(3600000, 900000)
```

Parameters

Name	Type	Description	Required
adjustedServerTimeInMs	Integer	The expected server time in milliseconds accounting for time difference between the client and server.	Yes

Name	Type	Description	Required
callStartTimeInMs	Integer	The time in milliseconds when the call starts.	Yes

Returns

`{String}` The elapsed time in MINUTES:SECONDS.

convertDateToISODateString(aDate)

Converts the date object to an ISO date string.

Example

```
finesse.utilities.Utilities.convertDateToISODateString(new Date())
```

Parameters

Name	Type	Description	Required
aDate	Date	The date to be converted to an ISO date format.	Yes

Returns

`{String}` The date in ISO format.

Note Certain browsers do not support the date constructor which considers ISO-8601 date format. For example, Internet Explorer 8.

convertNullToEmptyString(str)

Checks whether the string is null.

Example

```
finesse.utilities.Utilities.convertNullToEmptyString(null)
```

Parameters

Name	Type	Description	Required
str	String	The string object to be checked.	Yes

Returns

`{String}` The empty string if it is null or the string itself.

convertToServerTimeMillis(clientTime)

Converts the client time to the server time by adjusting time difference between server and client.

Example

```
convertToServerTimeMillis(new Date().getTime());
```

Parameters

Name	Type	Description	Required
clientTime	Integer	The client time in milliseconds.	Yes

Returns

`{Integer}` The server time in milliseconds.

convertTsToDuration(timestamp, now)

Converts the timestamp string of the format YYYY-MM-DDTHH:MM:SSZ into a duration of HH:MM:SS format.

Example

```
finesse.utilities.Utilities.convertTsToDuration(a)
```

Parameters

Name	Type	Description	Required
timestamp	String	The timestamp of the format YYYY-MM-DDTHH:MM:SSZ.	Yes
now	Date	The defined time from which to calculate the duration instead of using the current time.	Optional

Returns

`{String}` The duration in the HH:MM:SS format.

convertTsToDurationWithFormat(timestamp, forFormat, now)

Converts the timestamp string of the format YYYY-MM-DDTHH:MM:SSZ into a duration of HH:MM:SS format or -1 for null or negative durations, depending on the forFormat parameter.

Example

```
finesse.utilities.Utilities.convertTsToDurationWithFormat(a, true, new Date())
```

Parameters

Name	Type	Description	Required
timestamp	String	The timestamp of the format YYYY-MM-DDTHH:MM:SSZ.	Yes
forFormat	Boolean	Determines the timestamp format. • True—If the duration is null or a negative value, -1 is returned. Otherwise, the duration is in HH:MM:SS format. • False—The duration in HH:MM:SS format.	Yes
now	Date	The defined time from which to calculate the duration instead of using the current time.	Optional

Returns

`{String}` The duration in the HH:MM:SS format or -1 for null or negative values.

currentServerTimeMillis()

Retrieves the current time, which is adjusted by the calculated time difference to the approximate server time. This is used when calculating durations based on a server timestamp, which can produce unexpected results if the clock on the client and server are off.

Example

```
finesse.utilities.Utilities.currentServerTimeMillis()
```

Returns

`{String}` The current server time in milliseconds.

encodeNodeName(node)

Encodes the node name.

Example

```
finesse.utilities.Utilities.encodeNodeName('User1@h')
```

Parameters

Name	Type	Description	Required
node	String	The name of the node. If the string has special characters (?, @, &, ') then, the string is encoded in UTF-8.	Yes

Returns

`{String}` The encoded name of the node.

escapeSpaces(text)

Escapes the spaces as encoded " " characters so they can be safely rendered by jQuery.text(string) in all browsers. Although Internet Explorer behaves as expected, Firefox collapses spaces if this function is not used.

Example

```
finesse.utilities.Utilities.escapeSpaces('User John')
```

Parameters

Name	Type	Description	Required
text	String	The string whose spaces must be escaped.	Yes

Returns

`{String}` The string with spaces escaped.

extractHostname(url)

Extracts the hostname from a given URL string.

Example

`finesse.utilities.Utilities.extractHostname(a)`

Parameters

Name	Type	Description	Required
url	String	The URL from which the hostname of the server is extracted.	Yes

Returns

`{String}` The hostname of the server. For example, if the given URL is `https://finesse25.autobot.cvp:8445/desktop/container/?locale=en_US`, then the returned hostname will be “finesse25.autobot.cvp”.

extractTime(timeStr)

Extracts the time in milliseconds from the ISO date string.

Example

```
finesse.utilities.Utilities.extractTime('2020-05-25T13:49:42.80Z')
```

Parameters

Name	Type	Description	Required
timeStr	String	The time in ISO-8601 format (YYYY-MM-DDTHH:mm:ss.sssZ) or empty.	Yes

Returns

`{Long}` The number of milliseconds since 1 January 1970 (Unix Epoch). If the timeStr is empty, then the time returned is 0.

generateUUID()

Generates an RFC1422v4-compliant UUID using pseudorandom numbers.

Example

```
finesse.utilities.Utilities.generateUUID()
```

Returns

`{String}` An RFC1422v4-compliant UUID using pseudorandom numbers. For example, `456efbab-d794-4c7a-a731-762e476eb4d3`.

getAuthHeaderString(configObj)

Retrieves the authorization header that is based on SSO or non-SSO deployment. The headers are Bearer or Basic.

Example

```
finesse.utilities.Utilities.getAuthHeaderString(finesse.container.Config)
```

Parameters

Name	Type	Description	Required
configObj	Object	The configuration data which is either the `finesse.container.Config` or `finesse.gadget.Config`. For more information on gadget configuration, see Gadget Configuration, on page 492.	Yes

Returns

`{String}` The authorization header based on SSO or non-SSO deployment. For example,

SSO: Bearer MTAwMTAwMjpjaXNjbw==

NON-SSO: Basic MTAwMTAwMjpjaXNjbw==

getAuthModes()

Retrieves the constant for authentication modes. The modes are SSO, NON_SSO, or HYBRID.

Example

```
finesse.utilities.Utilities.getAuthModes()
```

Returns

`{String}` The constant for authentication modes, that is SSO, NON_SSO, or HYBRID.

getAuthTokenObj()

Retrieves the user access token as JSON Object.

Example

```
finesse.utilities.Utilities.getAuthTokenObj()
```

Returns

`{Object}` A user access token as the JSON object in SSO mode and null in non-SSO mode.

getCurrentDrift()

Retrieves the current timestamp difference between the client and server.

Example

```
finesse.utilities.Utilities.getCurrentDrift()
```

Returns

`{Integer}` The timestamp difference between the client and server. If it cannot be calculated, then it returns 0.

getDisplayTime(time)

Retrieves the timestamp value from milliseconds to the HH:MM:SS format.

Example

```
finesse.utilities.Utilities.getDisplayTime(60000)
```

Parameters

Name	Type	Description	Required
time	Number	The timestamp in milliseconds.	Yes

Returns

`{String}` The time string in the HH:MM:SS format.

getEquals(obj1, obj2)

Retrieves the value by comparing the value of each key in the first object with the value of the same key in the second object.

Example

```
finesse.utilities.Utilities.getEquals(x,y)
```

Parameters

Name	Type	Description	Required
obj1	Object	The first object to compare from. For example, x={'a': 1, 'b': 2}.	Yes
obj2	Object	The second object to compare against. For example, y={'b': 2, 'a': 1}.	Yes

Returns

`{Boolean}` True if the value of each key in the first object matches the value of the same key in the second object. False, if the value of at least one key in the first object does not match the value of the same key in the second object.

getParameterByName(str, name)

Accepts the value from the corresponding given string.

Example

```
finesse.utilities.Utilities.getParameterByName('http://www.company.com/?param1=value1&param2=value2',
 'param1')
```

Parameters

Name	Type	Description	Required
str	String	The string to search from. The URL from which parameter value is extracted based on the given parameter name.	Yes
name	String	The name to search for. The name of the parameter whose value must be extracted from the given URL string.	Yes

Returns

`{String}` The value that corresponds to the given name.

getQueryString(field, url)

Retrieves the value of the given field from the query string.

Example

```
finesse.utilities.Utilities.getQueryString('name', 'https://finesse/?name=user');
```

Parameters

Name	Type	Description	Required
field	String	The name of the field in the query string to retrieve.	Yes
url	String	The URL value from the query string to retrieve.	Optional

Returns

`{String}` The field value.

getToken()

Retrieves the user access token as a string.

Example

```
finesse.utilities.Utilities.getToken()
```

Returns

`{String}` The access token.

getUserAuthString()

Retrieves the Base64 encoded user authorization string.

Example

```
finesse.utilities.Utilities.getUserAuthString()
```

Returns

`{String}` The authorization string.

isArray(obj)

Determines whether an object is an array.

Example

```
finesse.utilities.Utilities.isArray(a)
```

Parameters

Name	Type	Description	Required
obj	Object	The object to be tested.	Yes

Returns

`{Boolean}` True if the object is an array, else false.

parseDateStringISO8601(s)

Parses the string with ISO-8601 date format: YYYY-MM-DDTHH:mm:ss.sssZ.

Example

```
finesse.utilities.Utilities.parseDateStringISO8601(new Date().toISOString())
```

Parameters

Name	Type	Description	Required
s	String	ISO-8601 date format: YYYY-MM-DDTHH:mm:ss.sssZ.	Yes

Returns

`{Date}` The JavaScript date object.

Note Certain browsers do not support the date constructor which considers ISO-8601 date format. For example, Internet Explorer 8.

removeSpaces(string)

Removes all spaces from the given string.

Example

```
finesse.utilities.Utilities.removeSpaces('user is')
```

Parameters

Name	Type	Description	Required
string	String	The string to remove spaces from.	Yes

Returns

`{String}` The string with no leading and trailing whitespace.

trim(str)

Trims the leading and trailing whitespace from the given string.

Example

```
finesse.utilities.Utilities.trim('user ')
```

Parameters

Name	Type	Description	Required
str	String	The string to trim.	Yes

Returns

`{String}` The string with removed whitespace from both ends.

validateHandler(handler)

Checks whether the given handler is a function.

Example

```
finesse.utilities.Utilities.validateHandler(a);
```

Parameters

Name	Type	Description	Required
handler	Function	Any valid JavaScript function.	Yes

Throws

`{Error}` If the handler provided is invalid, or generic JavaScript error stating `handler must be a function`.

Returns

`{Function}` The provided handler if it is valid.

whenAllDone()

Uses jQuery implementation of Promises (Deferred) to run the code when multiple asynchronous processes have completed.

Example

```
var asyncProcess1 = $.Deferred(),
    asyncProcess2 = $.Deferred();

finesse.utilities.Utilities.whenAllDone(asyncProcess1, asyncProcess2) // WHEN both
asyncProcess1 and asyncProcess2 are resolved or rejected ...
    .then(
       // First function passed to then() is called when all async processes are complete,
 regardless of errors
        function() {
            // Perform Logic("all processes completed");
        },
        // Second function passed to then() is called if any async processed threw an
exception
        function(failures) { // Array of failure messages
            // Perform Logic("Number of failed async processes: " + failures.length);
        });
```

Returns

`{Object}` A jQuery deferred object. For more information, see https://api.jquery.com/jQuery.Deferred/.

Desktop Cache

Class finesse.utilities.DesktopCache

Allows gadgets to cache their data locally in the IndexedDB of the browser. When a third-party gadget makes any REST API call to retrieve data from the server (which does not change frequently), it can be cached using Desktop Cache. This helps to load the gadget faster after failover by avoiding a REST request to the server. The cached data is a key-value pair where the key is a string, and value is an object.

Methods

clearData(callback)

Clears all the records in the database.

Example

```
finesse.utilities.DesktopCache.clearData(function(err) {
    if (!err) {
        // Successfully cleared the records!
    }
});
```

Parameters

Name	Type	Description	Required
callback	Function	An asynchronous callback function that is invoked after all the records in the IndexedDB is cleared	No

deleteData(key, callback)

Deletes the record that corresponds to the key passed from the database.

Example

```
// inside the gadget somewhere
if (userClickedOk) {
    finesse.utilities.DesktopCache.delete('someKey', function(err, data) {
        if (!err) {
            // Successfully deleted! Now do something else.
        }
    });
}
```

Parameters

Name	Type	Description	Required
key	String	The unique identifier which is used to delete a record from the IndexedDB.	Yes
callback	Function	An asynchronous callback function that is invoked after the attempt to delete the record (success or fail).	No
-->err	Object	The JavaScript error message.	No

fetchData(key, callback)

Retrieves the records corresponding to the key passed.

Example

```
 // fetching all the records
 finesse.utilities.DesktopCache.fetchData(null, function(err, data) {
     if (!err)
         console.log(data); // will print something like [{key: someKey, data: someData},
{key: otherKey, data: otherData}.......]
 });
```

Parameters

Name	Type	Description	Required
key	String	Unique identifier and it is the primary key to retrieve a single record from the IndexedDB.	Yes
callback	Function	An asynchronous callback function that is invoked after the data retrieval is successful or failed.	No
-->err	Object	The JavaScript error message.	No
-->data	Object	The JavaScript array of objects.	No

saveOrUpdateData(records, callback)

Creates or updates the records. The parameters passed are an array of key-value pairs.

Example

```
finesse.utilities.DesktopCache.saveOrUpdateData({
    [
        key: 'someKey'
        data: 'someData'
    ]
}, function(err, data) {
    if (!err) // do something
});
```

Parameters

Name	Type	Description	Required
records	Array	The data to be saved or updated as an array of key-value pairs.	Yes
callback	Function	An asynchronous callback function that is invoked after the save or updating of the record is completed (success or fail).	No
-->err	Object	The JavaScript error message.	No

setGroup(groupName)

Reduces redundancy in the data that are stored when different gadgets from the same vendor (for example, cuic) retain the same group name.

Note Having the same group name for the gadgets which share the same data reduces the server load and improves the performance. This API must be called before any other desktop cache APIs.

For example, consider vendor1 has two gadgets (gadget1 and gadget2), and both gadgets require the same token to be accessed from the server. Then the group name for these two gadgets can be set as vendor1. The gadget which loads first makes the server call, fetches the token, and then saves it in the desktop cache. The second gadget fetches this token from the desktop cache without making the server call.

Example

```
// Set this once and you don't need to send this group name with any other subsequent
requests from this gadget.
finesse.utilities.DesktopCache.setGroup('cuic');
```

Parameters

Name	Type	Description	Required
groupName	String	The name of the group for the gadgets which share the same data.	Yes

JSONValidator

Class finesse.utilities.JsonValidator

Utility methods for the validation of JSON data against a user-provided schema.

Methods

validateJson(jsonData, schema)

Validates the JSON data by applying a specific schema.

Parameters

Name	Type	Description	Required
jsonData	jsonData	The JSON data to be validated.	Yes
schema	schema	The JSON schema that validates the parameter jsonData. Follow the JSON schema definition standards. For more information, see http://json-schema.org/.	Yes

Returns

`{String}` The JSON data in the following format:

```
 {
    "valid": [true / false],
    "error": [tv4 error object
        if schema is not valid
    ]
}
```

The error object is as follows:

```
{
    "code": 0,
    "message": "Invalid type: string",
    "dataPath": "/intKey",
    "schemaPath": "/properties/intKey/type"
}
```

WorkflowService

Class finesse.workflow.WorkflowService

Provides an API which consists of methods that allow a gadget to submit the workflow task.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
workflowService = finesse.workflow.WorkflowService.init(containerServices);
var payload = {
    "dialogId": "email1",
    "mediaType": "email",
    "state": "EMAIL_READ",
    "taskVariables": {
        "from": "mme@cisco.com",
        "cc": "yyy@cisco.com"
    }
}
workflowService.submitTask(payload);
```

Methods

init(containerServices)

Initiates the WorkflowService and accepts finesse.containerservices.ContainerServices as a parameter.

Example

```
var containerServices = finesse.containerservices.ContainerServices.init();
workflowService = finesse.workflow.WorkflowService.init(containerServices);
```

Parameters

Name	Type	Description	Required
ContainerServices	Function	Provides container level services for gadget developers. Gadgets can utilize the container dialogs and event handling to add or remove a service.	Yes

submitTask(payload)

Allows to trigger workflow for digital channels.

Example

```
var payload = {
    "dialogId": "email1",
    "mediaType": "email",
    "state": "EMAIL_READ",
    "taskVariables": {
        "from": "mme@cisco.com",
        "cc": "yyy@cisco.com"
    }
}
workflowService.submitTask(payload);
```

Parameters

Name	Type	Description	Required
payload	Object	The action data of the JSON object as per the specification.	Yes

Name	Type	Description	Required
-->dialogId	String	Unique identifier of the dialog that helps in debugging an error. For example, dialogId, eventId, chatId and so on.	Yes
-->mediaType	String	The type of media under which the dialog is classified. For example, CHAT and EMAIL.	Yes
-->state	Enum	The workflow statuses are based on the task that is run. Values for Unified CCX are: • CHAT • CHAT_PRESENTED • CHAT_ACCEPTED • CHAT_HANDLED • CHAT_DECLINED • CHAT_LEAVE • EMAIL • EMAIL_PRESENTED • EMAIL_READ • EMAIL_DISCARDED • EMAIL_REPLIED • EMAIL_FORWARDED • EMAIL_REQUEUED Values for Unified CCE are: • TASK_OFFERED • TASK_ACCEPTED • TASK_ACTIVE • TASK_PAUSED • TASK_INTERRUPTED • TASK_CLOSED	Yes
-->taskVariables	Array	The corresponding value of the workflow name for the individual task. For example, the keys of the callVariables are callVariable name and callVariable value.	No

JSON Schema

JSON schema is a powerful tool for validating the structure of JSON data. The following are the advantages of using JSON schema.

- Describes your existing data formats.
- Provides clear human-and machine-readable documentation.
- Validates data that is useful for:
  - Automated testing.
  - Ensuring quality of client-submitted data.

The most basic schema is a blank JSON object, which constrains nothing, allows anything, and describes nothing.

Example

```
{ }
```

You can apply constraints on an instance by adding validation keywords to the schema. Consider the *type* keyword which can be used to restrict an instance to an object, array, string, number, boolean, or null.

Example

```
{ "type": "string" }
```

Certain APIs in Finesse JavaScript library may require the user to input objects with complex structures. Having a standard way of validating and providing the input is beneficial for both the consumer and the provider.

There are classes such as the **PopoverSchema** and **ChannelSchema** in Finesse JavaScript library which provide APIs such as the `getActionDataSchema()` and `getChannelConfigSchema()`. These APIs return a human-readable schema for a valid input for the APIs such as the `showPopover()` and `addChannel()` in a human-readable fashion.

Inspecting the JSON schemas helps to realize the combinations of valid inputs, which can be provided to some of the Finesse JavaScript APIs. For more information on understanding the JSON schema, see https://json-schema.org/understanding-json-schema/.

CHAPTER 11

Log Collection

Log Collection

These commands prompt you to specify a secure FTP (SFTP) server location to which the files will be uploaded.

To obtain logs:

- Install log: **file get install desktop-install.log**

  Use this command to see the installation log after the system is installed.

  This log is written to the SFTP server and stored as a text file written to this path: *<IP Address>\<date time stamp>\install\desktop-install.log*

- Desktop logs: **file get activelog desktop recurs compress**

  Use this command to obtain logs for the Finesse web applications. This command uploads a zip file that contains the following directories:

  - **webservices:** Contains the logs for the Finesse backend that serves the Finesse REST APIs. The maximum size of an uncompressed desktop log file is 100 MB. The maximum size of this directory is approximately 4.5 GB. After a log file reaches 100 MB, that file is compressed and a new log file is generated. Output to the last compressed desktop log file wraps to the log file created next. The log file wrap-up duration can vary, based on the number of users on the system. Timestamps are placed in the file name of each desktop log.

  - **desktop:** Contains logs from the Finesse agent desktop gadget container that holds the Finesse desktop gadgets. Any container-level errors with Finesse agent desktop will appear in these log files.

  - **admin:** Contains logs from the Finesse administration gadget container that holds the administration gadgets. Any container-level errors with the Finesse administration console appear in these log files.

    - **audit-log:** Audit logs contain all admin operations (including Finesse admin UI and REST client operations) and supervisor operations for Team Message. The maximum size of an uncompressed audit log file is 100 MB. The maximum size of total audit log files (including compressed log files) is approximately 1 GB. After a log file reaches 100 MB, that file is compressed and a new log file is generated. The log file wrap-up duration can vary, based on the number of users on the system. The log contains the following parameters:

    - Timestamp
    - User Id of the administrator
    - Method of operation (PUT, POST, DELETE). GET operations will not be logged
    - URL
    - Payload

  - **clientlogs:** Contains the client-side logs that are submitted from the Cisco Finesse agent desktop to the Finesse server. Each log file is no larger than 1.5 MB and contains a timestamp and the agent ID of the agent who submitted the file. A new log file is created each time that an agent submits client-side logs (the data is not appended to an existing log file). The maximum size of this directory is 100 MB. The directory holds a maximum number of 25000 clientlog files. When the directory exceeds the size limit or the file count, the oldest files are deleted.
  - **openfireservice:** Contains startup and shutdown-related information logs for the Cisco Finesse Notification Service.
  - **openfire:** Contains limited error and information logs for the Cisco Finesse Notification Service.
  - **finesse-dp:** Contains startup, error, and information logs generated by the Finesse Diagnostic Portal application.
  - **realm:** Contains the logs for authentication requests from clients that are handled by the Finesse backend.
  - **db:** Contains the Finesse database logs.
  - **/finesse/logs:** Contains the logs for the Cisco Finesse Tomcat service.
  - **fippa:** Contains logs for the Finesse IP Phone Agent (IPPA) application.
  - **3rdpartygadget:** Contains information, error, startup, and shutdown-related logs for the Cisco Finesse 3rdpartygadget server.
  - **jmx:** Contains the JMX counters data that is generated by the JMX logger process. It contains important jmx counters that are exposed by Finesse and openfire.
  - **finesse_maintenance_mode.log:** Contains the logs of Cisco Finesse hook script implementation of orchestration manager.

  These logs are stored in the following path on the SFTP server: *<IP address>\<date time stamp>\active_nnn.tgz* , where `nnn` is timestamp in long format.

- WebProxy Service logs: **file get activelog webproxy recurs compress**

  Use this command to obtain logs for the WebProxy Service. The maximum size of an uncompressed webproxy log file is 10 MB. The maximum size of this directory is approximately 500 MB. After a log file reaches 10 MB, that file is compressed and wraps to the new log file which is generated. The log file wrap-up duration can vary, based on the number of users on the system. Timestamps are placed in the file name of each webproxy log.

  These logs are stored in the following path on the SFTP server: *<IP address>\<date time stamp>\active_nnn.tgz* , where `nnn` is timestamp in long format.

  This command uploads a zip file that contains the following log files:

- **access.log**: Contains the webproxy access logs after you configure the access log-level using the **set webproxy access-log-level** CLI. For more information on CLI commands, see *WebProxy Service*.
- **error.log**: Contains the webproxy error logs.
- **webproxy_cli.log**: Contains the webproxy CLI logs. For more information on CLI commands, see *WebProxy Service*.
- **webproxy_launcher.log**: Contains the logs after the WebProxy Service is launched.

Note To access the individual log file, use the command **file get activelog webproxy/<log filename>**.

For example, **file get activelog webproxy/error.log**

- Servm log: **file get activelog platform/log/servm*.* compress**

  Use this command to obtain logs that are generated by the platform service manager that manages the starting and stopping of the Finesse services.

  The desktop and servm logs are compressed to one set of files.

  These logs are stored to the following path on the SFTP server: *<IP address>\<date time stamp>\active_nnn.tgz* , where `nnn` is the timestamp in long format.

- Platform Tomcat logs: **file get activelog tomcat/logs recurs compress**

  These logs are stored to the following path on the SFTP server: *<IP address>\<date time stamp>\active_nnn.tgz* , where `nnn` is the timestamp in long format.

- Install log: **file get install install.log**

  These logs are stored to the following path on the SFTP server: *<IP address>\<date time stamp>\active_nnn.tgz* , where `nnn` is timestamp in long format.

Note Log collection may fail when you use the compress flag if there are a lot of log files. If collection fails, run the command again without the compress flag.

Call Variables Logging

From Cisco Finesse Release 12.5(1) onwards, the call variables logging in Cisco Finesse logs are disabled by default. The callVariables contain sensitive user information and this property allows the administrator to decide whether the information must be captured in the logs. You can enable the call variables logging by using the CLI commands.

CHAPTER 12

Documents and Documentation Feedback

Documents and Documentation Feedback

Documents

The *Cisco Finesse Web Services Developer Guide* is available from Cisco DevNet at the following link:

https://developer.cisco.com/site/finesse/

If you have development questions, you can post them to the Cisco Finesse forums on Cisco DevNet, located at the following link: https://communities.cisco.com/community/developer/finesse.

The following documents are available from the Finesse page on Cisco.com (http://www.cisco.com/en/US/products/ps11324/tsd_products_support_series_home.html):

- *Cisco Finesse Installation and Upgrade Guide*
- *Cisco Finesse Administration Guide*
- *Release Notes for Cisco Finesse*

JavaScript Library and Sample Gadgets

The Finesse JavaScript library and sample gadgets are available on Cisco DevNet at the following link: https://developer.cisco.com/site/finesse/

Documentation Feedback

You can provide comments about this document by sending email to the following address: contactcenterproducts_docfeedback@cisco.com

We appreciate your comments.