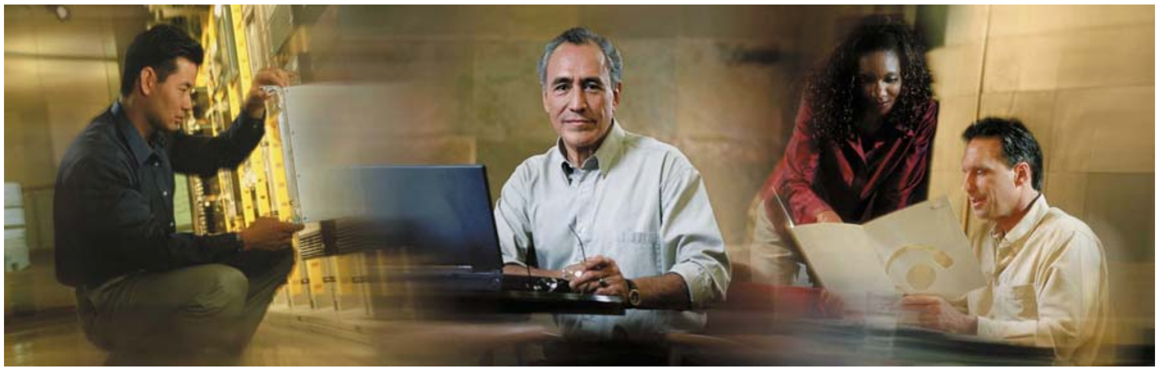


Cisco Collaboration Server Version 5.0

Client-side API Programmer's Guide

Table Of Contents

Introduction To This Guide

This document describes the Cisco Collaboration Server Client-side Application Programming Interface (API). The API allows you to create an agent's client-interface to interact with the Collaboration Server.

This section contains the following:

- Audience
- Conventions
- Online And PDF Versions Of This Guide
- About This Guide
- Supported Browser Versions
- Obtaining Technical Support

Audience

This guide provides instructions, reference information, and examples for programmers and developers creating client applications that interface with Cisco Collaboration Server.

It is recommended that users of this guide also read the *Cisco Collaboration Server Administration Guide*.

This guide assumes you have experience writing Javascript applications, as well as an understanding of programming event-driven applications.

Conventions

This guide uses the convention `/Cisco_CS/path/to/file` when discussing the path to various files. Depending on your installation and operating system the actual path may be slightly different. Windows installation defaults to `C:\Cisco_CS\`. Solaris installations default to the point at which you run the install script. The name Cisco_CS is the default Collaboration Server root directory. This name can be changed by the installer of the program. Even though the name of the root directory may be different, the following paths described in this document will be the same regardless of your installation.

Online and PDF Versions Of This Guide

This guide is available in an online version (HTML) and a PDF version. The Online version contains cross-reference links between all of the functions and events. The PDF version should be used for printing a hard copy of this document. Both versions contain identical text.

About This Guide

This guide contains the following sections:

- Overview - an overview on how the API works.
- Creating a Simple Chat Applications - a short tutorial on creating a simple chat application using the API.
- Included Sample Applications - Descriptions of the two sample applications that are included with the API.
- Functions - The complete list of functions available in the API with events and examples.
- Events - The complete list of events available in the API with examples.
- Event Handler Files - The code listings of the event handler files.

Supported Browser Versions

The following browsers are supported for use with the API Client-side application that you create:

Supported operating systems for Collaboration agents	**Supported agent browsers**			
	Internet Explorer 5.01 SP2	**Internet Explorer 5.5 SP2**	**Internet Explorer 6.0**	**Netscape Navigator 4.78**
Windows NT 4.0 SP6A	Supported	Supported	Supported	Supported
Windows 2000	Supported	Supported	Supported	

Windows XP	Supported

Caution: When you are using Netscape Navigator, you should not attempt to resize the browser window during an active session. Doing so causes the agent applet to reload, which can cause unpredictable behavior.

Browser Settings

Before using your application, ensure that your browser is set up to:

- Always accept cookies.
- Compare the document in cache to the document on the network once per session.

For specific instructions on how to change these settings, see the documentation for your browser.

Obtaining Technical Support

If you have questions or problems, call the Cisco Systems Internet Communications Software Group (ICSG) Technical Assistance Center at (800) 553-2447, or send email to tac@cisco.com. Customer's calling outside the US can dial (408) 526-7209 or visit the TAC Web site at http://www.cisco.com/public/support/tac/home.shtml for additional worldwide contact numbers.

Getting Started with the API

This section contains the following:

- About The API
- Features
- The API Files
- apiAgents.properties

About The API

The Collaboration Server Client-side API allows application programmers to create feature-limited client-applications of Collaboration Server.

You can use the Client-side API to integrate parts of Collaborations Server's Single-session Agent Desktop into your own web-based application. For example, you can add chat and page sharing to an existing agent application that is browser based.

The API uses standard Javascript calls that are supported in the latest versions of the Netscape Navigator and Internet Explorer. Any application that you create that runs in these browsers can be further enhanced by adding the functionality provided by the Collaboration Server Client-side API.

Features

The following tables lists the features that are supported in this version of the Collaboration Server Client-side API:

Feature	**Description**
Single-Session Chat	Text-based chat between the caller and the agent.
PageShare	Share a web page with the other participant.
FormShare	Share a form with the other participant in the session.

PagePush	Send a URL to the other participant's shared browser.

Features not listed in this table may work with the Client-side API, but they are not supported by Cisco.

The API Files

The Cisco Collaboration Server Client-side API consists of these base files:

- API.jhtml - The API file. This file should NOT be edited. This file contains a polling applets that receives events from the Collaboration Server.
- APIHandler.js - Javascript event handlers. You must modify these functions to perform according to your specific application's needs.
- HiddenFrame.html - An html file that must be included in your web application. It is used as the required target for server responses. This file should NOT be edited.

In addition, there are also two other files. These files are used for creating a separate authentication application that then links to your main application.

- APILite.jhtml - Contains only the authentication functions in the API.
- APILiteHandler.js - Contains only the authentication events in the API. This file does not contain the polling applet required to receive events from the Collaboration Server.

apiAgent.properties

This file specifies some properties that are used for agents who are logged into the Collaboration Server through the API. This file is located in **`/Cisco_CS/servlet/properties/`**

Property	**Description**
wrapUpMode **Note:** This setting overrides the agent's Wrap Up setting in the agents Role.	0 - No Wrap Up 1- Automatic Wrap Up 2 - Manual Wrap Up - Default wrap up at session end. 3 - Manual Wrap Up - Default OFF
linkDownTimeUp	The amount of time, in milliseconds, before

	a LinkDownEvent is generated by the polling applet because the application cannot communicate with the server.

Creating an Application

This section provides an overview of using the API to create an application.

This section contains the following:

- Two Methods Of Creating An Application
- Calling An API Function From Within An HTML Page
- Using Event Handlers
- Function Sequence in a Typical Collaborative Session
- Limitations

Two Methods of Creating an Application

There are two ways you can use the API to create an application:

- Method 1 uses the "lite" versions of the API files (ccsLite.jthml and APILiteHandler.js) for login and the full API files (css.jhtml & APIHandler.js) version for all other events - see Sample1.
- Method 2 uses the normal API file (API.jhtml & APIHandler.js) - see Sample2.

Calling an API Function From Within An HTML Page

The API functions are called using standard Javascript function calls. You call an API function from within HTML pages by use of *form* tags or *href* tags, for example:

```
<form name="LoginForm"
action="javascript:top.actionFrame.login(window.document.LoginForm.usern
ame.value,
 window.document.LoginForm.password.value)">
```

```
<A
HREF=javascript:top.actionFrame.login(window.document.LoginForm.username
.value,
 window.document.LoginForm.password.value)>Submit</A>
```

Use the method best suited to your application.

Using Event Handlers

For almost every action you take by using one of the API functions, the Collaboration Server returns an event. For example, if you attempt to login an agent using login(), then the Collaboration Server will return an event informing you whether the login was successful, (AgentLoginOkEvent), or if there was an error (InvalidAgentEvent).

The API does not define what happens in the event handlers. Collaboration Server calls the event handlers as required, but if you do not modify the event handler to do something, then nothing happens.

This guide provides some sample events for each of the event handlers, but you must customize the events for your own application.

Function Sequence in a Typical Collaborative Session

The table below outlines the functions that need to be used during a typical collaboration session. See the functions for the corresponding events that occur.

Note: You do not have to wait for a response from the server before calling another function. The server is able to receive additional requests while it is still processing the current request.

	Function	**Description**
1	login()	You must have the agent login before anything else can happen.
2	startEventPolling()	Once the agent is logged in you must start the event polling applet.
3	bind()	If you are going to share pages or forms you must bind your application to a browser window.
4	getAgentProperties()	Depending on what you want to set for visual indications of the various agent options, you can use getAgentProperties to determine the state of many of the options.
5	setReady()	Before the agent can collaborate they must be setReady("true").

Caller is Pushed to Agent		Once the agent is ready a caller waiting in the agent's skill queue will join into a session with the agent. The agent ready state is automatically set to false once the agent and caller are in session.
6	sendChat()	Once the agent is in session with a caller you can use the various session commands to enable collaborative functionality between the two parties.
	pageShare()	
	formShare()	
	pushPage()	
	followMe()	
	getChatHistory()	
7	disconnect()	The agent can disconnect() or the caller may hang up. In either case a UserDisconnectEvent is generated
8	startWrapUp()	At this point, if the agent is enabled for wrap on, but it is not on by default, then you must provide the agent with the ability to startWrapUp. This function can be called only after receiving a wrapUpNotifyEvent.
9	endWrapUp()	If wrap up was started then you can end it or cancel it.
	cancelWrapUp()	
10	logout() or setReady()	Once wrap up (if it was enabled) has been ended, then the agent can logout. If the agent will participate in another session, you must have their ready state set to *true*, and the process restarts.
NOTE	A LinkDownEvent is generated if the connection between the client and the server has been lost.	

Limitations

There are a few limitations to creating and using the API:

- When you log in an agent it should always be done in a "fresh" window. If you run into unusual behavior you should clear the cache and close the browser after you log out an agent.

- When testing, do not start an agent and a caller on the same computer using the same type of browser. The JVM is shared for the browsers and unpredictable behavior may occur. Instead, use two different browser types (i.e. log the agent in using Internet Explorer and log the caller in using Netscape Navigator), or use two different computers.

- Your application's main html page must not be resizeable in Netscape. Resizing the browser in Netscape causes applets to reload and could cause unpredictable behavior.

- Microsoft Internet Explorer users must use an external browser window to share complex pages or forms. You cannot share complex pages or forms in a shared frame.

- AgentLoginOkEvent and AgentReconnectEvent are the only events that can be received before startEventPolling() is called. All other events can only be received if startEventPolling() is running.

- Agents who use applications built from the API must be push agents. In addition, this API does not support Pick agents, custom agents, and MeetMe sessions.

Creating A Simple Chat Application

The topic provides a brief tutorial on using the Collaboration Server Client-side API. The tutorial leads you through the steps to create a simple chat application that can be used to chat with a caller.

Complete the following steps to complete a simple chat application:

1. Create an Agent to Test the API
2. Copy the API Files
3. Create the Required Files
4. Modify APIHandler.js
5. Login and Test the Chat Application

Create an Agent to Test the API

Before you begin you should create an agent and a role that will be used specifically for tutorial. You create the agent using the Collaboration Server Administration Desktop application (login in at http://<YOUR COLLABORATION SERVER>/admin..

- The agent should have a single-session skill assigned.
- For the purposes of this tutorial, you should set wrapUpMode=0 in `/Cisco_CS/servlet/properties/apiAgent.properties`, since this tutorial does not provide any Wrap Up controls.
- The agent must be a *push* agent, which is set in the agent role using the Collaboration Server Administration Desktop.

Copy the API files

First you need to copy some files from the API folder in /Cisco_CS/pub/html/api. You should copy the files to a new working directory that is accessible from your web server. The files that you should copy are:

- API.jhtml - DO NOT EDIT.
- HiddenFrame.html - DO NOT EDIT.

- APIHandler.js - Edit this file to modify the event handlers for your application. Suggested functions for this application are included below.

Create the Required Files

Next you need to create several html files. These files hold the framework of your chat application. and should be created in the same directory into which you copied the three files above.

The files that you need to create are:

- index.html - A frameset that loads the following frames.
- loginFrame.html - a frame that contains a form so that the agent can login to the Collaboration Server, set ready, disconnect, etc..
- chatReceiveFrame.html - a frame used to display chat received from the caller.
- chatSendFrame.html - a frame used to send chat to the caller.

The content of the files are shown below.

index.html

This file is the frameset that houses all of the other files in your applications. Take note of the top two frames in the frameset, hiddenFrame and actionFrame. These two frames are required in every application that uses the Collaboration Server Client-side API. They must be called by these names.

Also note that the frame "actionFrame" contains API.jhtml. API.jhtml contains all of the API functions, it also references APIHandler.js.

The remaining frames in the frameset are:

- the login frame, which contains the login fields as well as the logout, disconnect, and setReady forms,
- and the chatSendFrame and chatReceiveFrame, that are used to send and receive chat.

index.html

```
<html>
```

```
<head>

<title>A Simple Chat Application</title>

<frameset  rows="1,1,175,*">

<frame name="hiddenFrame" src="HiddenFrame.html" marginwidth="0"
marginheight="0" scrolling="no">

<frame name="actionFrame" src="API.jhtml" scrolling="no" noresize>

<frame name="loginFrame" src="loginFrame.html" scrolling="no" noresize>

<frameset  cols="50%,*">

<frame name="chatSendFrame" src="chatSendFrame.html" scrolling="no">

<frame name="chatReceiveFrame" src="chatReceiveFrame.html"
scrolling="auto">

</frameset>

</frameset>

</head>

<body>

You need frames to view this document.

</body>

</html>
```

chatReceiveFrame.html

This file is used to display chat messages from the agent and the caller. The chat text is displayed here when ChatMessageEvent is fired.

chatReceiveFrame.html

```
<html>
```

```
<head>

<title>Chat Receive Frame</title>

</head>

<body>

<hr>

</body>

</html>
```

chatSendFrame.html

This file is used to send chat to the caller. The chat message entered by the agent is cleared after each message is sent by ChatMessageEvent. Note the function call in the *action* portion of the *form* tag.

chatSendFrame.html

```
<html>

<head>

<title>Chat Send Frame</title>

</head>

<body>

<form name="sendChatForm"
action="javascript:top.actionFrame.sendChat(window.document.sendChatForm.me
ssage.value)">

<textarea cols="50" rows="10" name="message"></textarea>

<br><input type="submit" value="Send Chat">

</form>
```

```
</body>

</html>
```

loginFrame.html

This file contains the basic controls for the agent, including:

- Login
- Set Ready
- Session Disconnect
- Logout

Each of the *form* elements on the page call a different function in the API.

loginFrame.html

```
<html>

<head>

<title>Login Frame</title>

</head>

<body>

<form name="LoginForm"
action="javascript:top.actionFrame.login(window.document.LoginForm.username
.value,window.document.LoginForm.password.value)">

<p>Login name: <input type="text" name="username"> Password: <input
type="password" name="password"> <input type="submit" value="Login">

</form>

<form name="setReadyForm"
action="javascript:top.actionFrame.setReady(window.document.setReadyForm.re
adyState.value)">

<select name="readyState">
```

```
<option value="true">true</option>

<option value="false">false</option>

</select>

<input type="submit" value="Set Ready">

</form>

<form name="disconnectForm"
action="javascript:top.actionFrame.disconnect()">

<input type="submit" value="Session Disconnect">

</form>

<form name="LogoutButton" action="javascript:top.actionFrame.logout()">

<input type="submit" value="Logout">

</form>

</body>

</html>
```

Modify to APIHandler.js

Finally you need to make some modifications to APIHandler.js. APIHandler.js is the main interface into the API.

At a minimum, you should modify the following events in APIHandler.js for this tutorial:

- AgentLoginOkEvent
- AgentLogoutEvent
- AgentReadyEvent
- AddSessionEvent
- ChatMessageEvent

- UserDisconnectEvent
- DropSessionEvent
- NoSessionEvent

AgentLoginOkEvent

This event starts the event polling applet and pops up an alert notifying them that they are now logged in.

AgentLoginOkEvent

```
function AgentLoginOkEvent()

{

top.actionFrame.startEventPolling();

alert('You are now logged in.');

}
```

AgentLogoutEvent

This event notifies the agent that they have logged out.

AgentLogoutEvent

```
function AgentLogoutEvent()

{

alert('You have been logged out. Please close your browser.');

}
```

AgentReadyEvent

This event notifies the agent that they are ready (or unavailable) to participate in a session with a caller.

AgentReadyEvent

```
function AgentReadyEvent(ready)

{

if(ready == true) {

alert('You are now ready to receive a caller.');

} else {

alert('You are now unavailable.');

}

}
```

AddSessionEvent

This event notifies the agent that they are about to go into session with a caller.

AddSessionEvent

```
function AddSessionEvent(sessionId,agentParticipantId,callerParticipantId,
callerParticipantName,requestId)

{

alert('You are about to enter a new session with ' + callerParticipantName
+ '.');

top.chatReceiveFrame.document.write('<h3> Now in session with ' +
callerParticipantName + '.</h3>');

}
```

ChatMessageEvent

This event occurs each time a chat message is sent. If the sender is the agent then this event first clears the text that was sent from the agent's send chat frame, then the event prints the message text in chatReceiveFrame.html.

If the sender is the caller, then the event simply writes the callers text to the chat application.

ChatMessageEvent

```
function ChatMessageEvent(sessionId, senderName, messageText, isAgent)

{

if(isAgent == true)

{

top.chatSendFrame.sendChatForm.message.value = "";

top.chatReceiveFrame.document.write("<br>" + senderName + ": " +
messageText);

}

else if(isAgent == false)

{

top.chatReceiveFrame.document.write("<br><b>"+ senderName + "</b>: "  +
messageText);

}

}
```

DropSessionEvent

This event notifies the agent that the caller has disconnected. It also prints a short message in chatReceiveFrame.html.

DropSessionEvent

```
function DropSessionEvent(sessionId, participantName)

{

alert('The session with, ' + participantName + ', has ended.');

top.chatReceiveFrame.document.write('<h3>' + participantName + ' has left
the session.</h3>');

}
```

NoSessionEvent

This event occurs if the agent tries to access a function that requires a session (such as send chat) while the agent is not currently in a session.

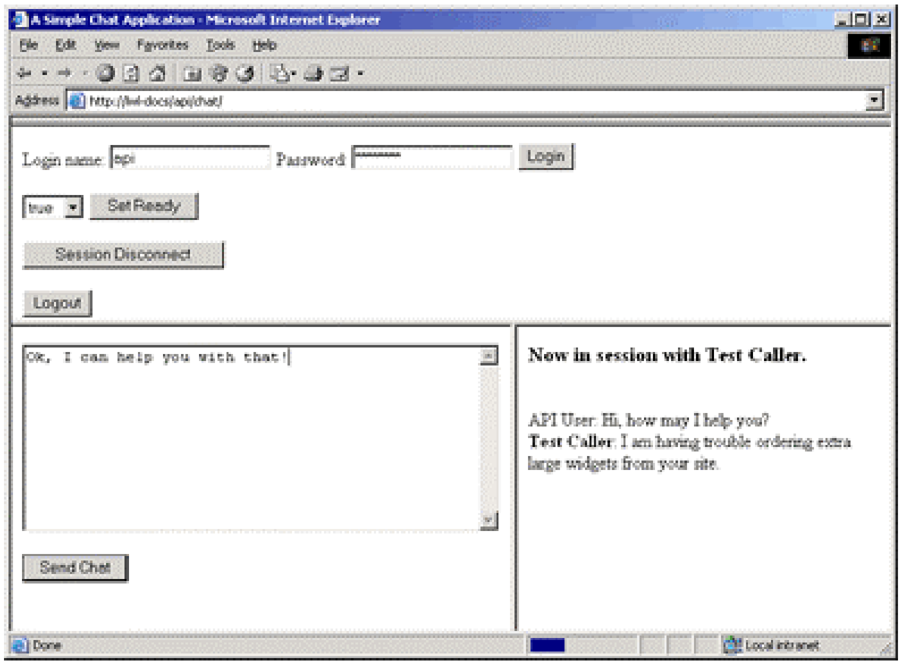
NoSessionEvent

```

function NoSessionEvent()

{

alert('You cannot do that because you are not in session!');

}
```

Login and Test the Chat Application

After you have copied, created, and modified the necessary files you should attempt to login to the page to test the chat.

Load the /path/to/index.html file into your browser. It should look similar to the image below:


A Simple Chat Application - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://lwl-docs/api/chat/
Login name: api
Password
Login
true
Set Ready
Session Disconnect
Logout
Send Chat
Done
Local intranet

Ok, I can help you with that!

Now in session with Test Caller.

API User: Hi, how may I help you?
Test Caller: I am having trouble ordering extra large widgets from your site.

To Test the Application:

6. Log in an appropriate agent. Enter the agent's login name and password and click **Login**.

7. Set the agent ready state to **true** by selecting true and clicking **Set Ready**.

8. Log in a caller from http://<COLLABORATION SERVER>/callme/callFrame.html. **You must use a separate computer to login the computer.** Be sure to select a skill that is usable by the agent that you have logged in.

9. The caller and agent enter a session. Use text chat to communicate between the two browsers.

Included Sample Applications

Sample 1

The sample1 application included in the API distribution is a mini implementation of the Cisco COllaboration Server Single-session Chat Interface. The sample1 application can be found in the /Cisco_CS/api/sample1 directory.

Before using the sample you should understand the following concepts:

- Files In This Sample
- Using the Sample
- How the Sample Works

Files In This Sample

The directory contains the following files:

File	Description
agent_loggedout_page.html	The page that the agent is redirected to after a logout.
agent_already_loggedin.html	The page that the agent is redirected to if they attempt to login while they are already running another logged in client.
API.jhtml	The main API file (do not edit).
APIHandler.js	The event handler functions that have been customized for this sample. You should look closely at this file to see how the different event handlers have been coded.
APILite.jhtml	The login API file that contains only the login functions (do not edit).
APILiteHandler.js	The corresponding "lite" event handler for APILite.jhtml.
chat_frame.html	Contains:

	• chatFrameLower.html • chatFrameUpper.html • chatFrameUpperLeft.html chatFrameUpperRight.html
no_longer_in_use.html	The page that is loaded in place of login.html after the agent logs in. This page tells the agent to close the window since it is no longer in use. mini_CCS has loaded by this time.
EmptySharedFrame.html	The internal frame that is used for PageShare. The application uses bind() to bind all page sharing to this frame.
HiddenFrame.html	The required file loaded in the hidden frame (HiddenFrame). This file is required in every application that you create using the Collaboration Server Client-side API.
invalid_password.html	The page that loads if the agent tries to login with an invalid password.
invalid_username.html	The page that loads if the agent tries to login with an invalid login name.
login.html	The page that is used by the agent to login to the Collaboration Server using this sample. This is a frame set that contains the following files: • HiddenFrame.html • APILite.jhtml loginForm.html
mini_CCS.html	The frame set that loads the main application used by the agent to participate in a collaborative session with a caller. See Figure 1 below.
session_frame.html	The page that displays the number of callers in session (0 or 1 in this release of the API) and the number of callers in the queue
toolbar_frame.html	The page that loads and manages the tool icons. This page contains Javascript which changes the icon depending on the state of the various features. For example, the ready light icon shows an illuminated red light when the agent is not ready, and shows an illuminated green light when the

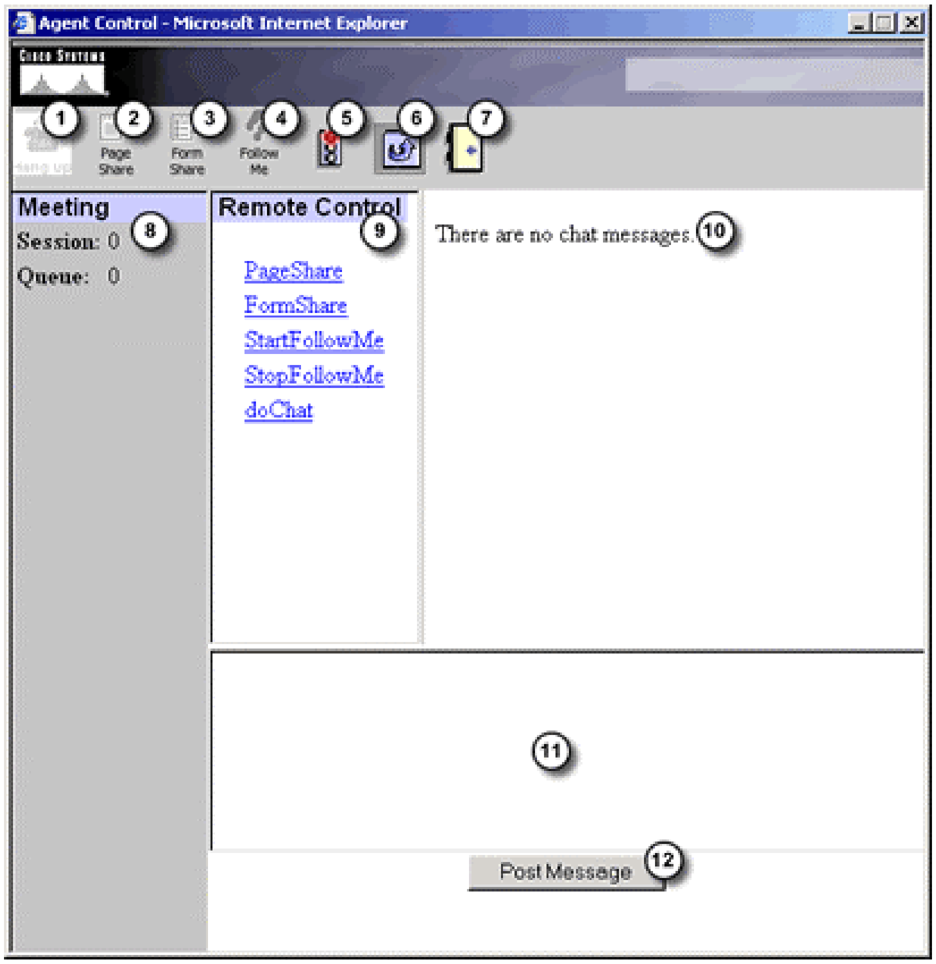
	agent is ready.
cisco_logo.html	The page loads the Cisco logo at the top of the page.

Using the Sample

Open http://<CCS SERVER/api/sample1/ login.html in a browser to use the sample1 application.

The page mini_CCS.html appears once the agent is logged in. See Figure 1.

Figure 1: miniCCS.html



Sample1 provides the following functionality (see callouts from Figure 1):

- Agent login (from sample1/login.html)

1. Session Disconnect - once in a session you can click **hangup** to end the session.
2. Page Share - The agent can click **PageShare** while in session to share the page currently in the agent's shared browser with the caller.
3. Form Share - The agent can click **FormShare** while in session to share the form currently in the agent's shared browser with the caller.
4. Follow Me - Toggles Follow Me browsing.
5. Ready/Not Ready Switch - This is a clickable switch that the agent uses to toggle his ready state. A red light indicates that the agent is not ready. A green light indicates that th agent is ready.
6. Wrap Up Mode - Starts/Stops wrap up mode if the agent is able to optionally use wrap up. If wrap up is on by default then the agent can use this button to end wrap up. If wrap up for the agent is disables then this button has no functionality.
7. Log Out - Clicking **Logout** logs the agent out of the application.
8. Session/Queue Caller Count - Displays the number of callers currently in session (0 or 1 in this version of he API), and also displays the number of callers currently in queue.
9. Caller Remote Control Panel - These items can be clicked to enable or disable the various features on the caller side.
10. Chat Message Display Area - Any text chat is shown in this frame.
11. Chat Message Input Area - The agent types chat into this box.
12. Chat Message Send - Clicking **Post Message** sends that chat in the Chat Message Input Area to the caller.

How the Sample Works

Notice that there are two API files included within this sample, API.jhtml and APILite.jhtml.

APILite.jhtml - (includes APILiteHandler.js) provides only the functions and events necessary for login and reconnect. These files are used by login.html, which is the agent login form. Once the agent login is authenticated then miniCCS.html is loaded.

API.jhtml - (includes ccsHander.js) Contains the full API, and is used by the mini-CCS application (miniCCS.html).

The agent logs in to the application using a page which calls APILite.jhtml (login.html). Once logged in the application loads a new window (miniCCS.html) that calls the complete API (API.jhtml).

Sample 2

The sample2 application included in the API distribution is a simple demonstration of the capabilities of the API.

Before using the sample you should understand the following concepts:

- Files In This Sample
- Using the Sample

Files In This Sample

The directory contains the following files:

File	Description
canvas_frame.html	The page used to print status messages into the top frame.
API.jhtml	The main API file (do not edit).
APIHandler.js	The event handler functions that have been customized for this sample. You should look closely at this file to see how the different event handlers have been coded.
EmptySharedFrame.html	This page is a placeholder in the frame is used to share web pages and forms. This page disappears when web pages are shared
HiddenFrame.html	The required file loaded in the hidden frame (hiddenFrame). This file is required in every application that you create using the Collaboration Server Client-side API.
main_form.html	This is the page that contains all of the function calls for the application
main_frame.html	This is the page that loads all of the other components. It is a simple frame set.

Using the Sample

Open http://<CCS SERVER>/api/sample2/main_frame.html in a browser to use the sample2 application.

The page contains a table that lists all of the functions used in the API. Some of the functions allow you to enter arguments (such as login()).

You should take a "top-down" approach when testing the functions on the page and test each function in sequence. At a minimum you should do the following in order:

1. login() - using an actual agent Login name and Password.
2. startEventPolling()
3. setReady() - Set to true.
4. bind()

Functions

bind()

This function allows you to bind a frame or a browser window to be used for PageShare. By using bind() you identify the frame or browser window into which the shared web pages should be viewed.

Name	bind
Description	Binds an internal or external page for sharing.
Arguments	frameName - String. The name of the internal frame to which you want to bind page sharing. If you want to use an external page then you can pass any name as frameName (for example, *null*). URL - String. The URL to load if you are using an external share page. Pass *null* if you are using an internal share page.
Usage	bind(frameName, URL)
Returns	none
Events	none
Requirements	Agent Must be logged in.

The simplest example of bind is shown below :

```
<A HREF=javascript:top.actionFrame.bind("emptySharedFrame",null)>Bind
Internal</A>
```

cancelWrapUp()

This function notifies the Collaboration Server that the agent is canceling wrap up. This function is used after a WrapUpNotifyEvent has been received.

Name	cancelWrapUp
Description	Notifies the Collaboration Server that the agent does not want to enter Wrap Up.
Arguments	none
Usage	cancelWrapUp()
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • WrapUpUnavailable - Wrap Up for this agent is not available because either this agent has not been enabled for wrap up in apiAgent.properties, or a WrapUpNotifyEvent has not been received. • UnableToCancelWrapUp - Unable to cancel Wrap Up due to an unspecified error. DropSessionEvent - The session is dropped.
Requirements	• Agent Must be logged in. • Agent must be in session. A WrapUpNotifyEvent must have been received.
Notes	When a WrapUpNotifyEvent is received, the Collaboration Server expects you to call either cancelWrapUp() or startWrapUp(). Until you do so the Collaboration Server logs the agent as still being in session with the caller, even though the caller may have exited from the session.

The simplest example of cancelWrapUp is shown below :

```
<form name="cancelWrapUpFrom"
action="javascript:top.actionFrame.cancelWrapUp()">


<input type="submit" value="Cancel Wrap Up">
```

```
</form>
```

disconnect()

This function disconnects the agent from the current session.

Note: This function does NOT log the agent out of the application. It only disconnects the agent from the session. You must use logout() to log the agent out of your application.

Name	disconnect
Description	Disconnects the user from the current session.
Arguments	none
Usage	disconnect()
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • NoSessionEvent - The agent is not currently in session. • UserDisconnectEvent - The user has successfully disconnected from the session. UnableToDisconnectEvent - Unable to disconnect from the session due to an unspecified error.
Requirements	• Agent Must be logged in. Agent must be in session.
Notes	A WrapUpNotifyEvent is generated if wrapUp is enabled.

The simplest example of disconnect is shown below :

```
<form name="disconnectForm"
action="javascript:top.actionFrame.disconnect()">

<input type="submit" value="Session Disconnect">

</form>
```

endWrapUp()

This function notifies the Collaboration Server that the agent has ended wrap up.

Name	endWrapUp
Description	Notifies the Collaboration Server that the agent has ended Wrap Up.
Arguments	none
Usage	endWrapUp()
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • WrapUpNotStarted - Wrap Up was not started for this agent and session. • WrapUpUnavailable - Wrap Up for this agent is not available because Wrap Up has not been started. • UnableToEndWrapUpEvent - Unable to end Wrap Up due to an unspecified error. DropSessionEvent - The session is dropped.
Requirements	• Agent Must be logged in. • Agent must be in session. Wrap Up must be started.
Notes	You must start wrap up using startWrapUp() before you can call this function.

The simplest example of endWrapUp is shown below :

```
<form name="cancelWrapUpFrom"
action="javascript:top.actionFrame.endWrapUp()">


<input type="submit" value="End Wrap Up">


</form>
```

followMe()

This function allows you to enable or disable followMe.

Name	followMe
Description	Enables or disables Follow Me browsing for the current shared browser.
Arguments	On - Boolean. *true* turns on Follow Me browsing. *false* turns off Follow Me browsing. Local - Boolean. *true* means apply to the local user. *false* means apply to the remote user.
Usage	formShare(On, Local)
Returns	none
Events	none
Requirements	• Agent Must be logged in. • Agent must be in session. Shared browser must be present.

The simplest example of followMe is shown below :

```
<form name="followMeForm"
action="javascript:top.actionFrame.followMe(window.document.followMeForm.lo
cal.value)">

<select name="local">

<option value="false">false</option>

<option value="true">true</option>

</select>

<input type="submit" value="Set Wrap Up">

</form>
```

formShare()

This function allows you to share forms between the caller and the agent.

Name	formShare
Description	Shares the page currently displayed in the shared browser
Arguments	Local - Boolean. *true* shares the form currently displayed in the local browser. *false* shares the form currently displayed in the remote browser.
Usage	formShare(Local)
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • NoSessionEvent - The agent is not currently in session. • FormOrPageShareEvent - Returns the web page URL being shared. UnableToSharePageEvent - The web page could not be shared due to an unspecified error.
Requirements	• Agent Must be logged in. • Agent must be in session. Shared browser must be present.
Notes	You must use bind() before calling this function.

The simplest example of formShare is shown below :

```
<form name="formShareForm"
action="javascript:top.actionFrame.formShare(window.document.formShareForm.
share.value)">


<select name="share">


<option value="false">false</option>
```

```
<option value="true">true</option>

</select>

<input type="submit" value="Share Form">

</form>
```

getAgentProperties()

getAgentProperties lets you retrieve individual agent properties and settings. The properties that can be retrieved are shown in the table below.

Name	getAgentProperties
Description	Gets a specific property from the list of available agent properties.
Arguments	Property Name-A String. The name of the property that you wish to retrieve.
Usage	getAgentProperties(Property Name);
Returns	none
Events	• AgentPropertyEvent - Returns the key and value of the requested property. • InvalidAgentProperty - Returns the key of the invalid property that was submitted to getAgentProperties(). AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server.
Requirements	Agent must be logged in.
Notes	The table below lists the properties that can be returned by getAgentProperties(). The keys are case sensitive.

Properties Returned from getAgentProperties():

Key	**Description**
fullName	The agent's full name (firstName followed by lastName).
firstName	The agent's first name.
lastName	The agent's last name.
loginName	The agent's login name.

loginDate	The date and time that the agent last logged in ithe format: *day, month, day of month, hh:mm:ss, time zone, and year*. For example "Thu Jan 10 15:49:58 EST 2002"
Skill	The agent's skills.
ApplStr	The agents Application String.
skillTargetID	The agent's unique ID of the agent's skill on the ICM. Available only for servers that are integrated with ICM.
peripheralID	The agent's unique ID of the peripheral that the agent or skill is assigned to on the ICM. Available only for servers that are integrated with ICM.
loginMediaClass	The agent's Login Media Class. Values can be: • SSC - Single-session Chat (no phones) • MSC - Multi-session Chat (no phones) • BC - Blended Collaboration but the agent is not ready for voice. • BC, SSC - Blended Collaboration & Single-session Chat but agent is not ready for voice. • BC, VOICE - Blended Collaboration and the agent is ready for voice. • BC, SSC, VOICE - Blended Collaboration & Single-session Chat and the agent is ready for Voice. VOICE - Used for agents assigned to an ACD. This type of agent is not supported using the API.
ACDTerminalID	The agent's ACD Terminal ID. Returns "" if not provisioned on the Collaboration Server.
isIPTA	Returns *true* or *false* depending on whether the agent provisioned for ICM Routing.
wrapUpEnabled	Returns *true* or *false* depending on whether the agent is enabled for Wrap up.
agentState	The states returned for the agentState property: • 1 - STATE_READY (The agent state when available). • 2 - STATE_READY (The agent state when available). • 3 - STATE_IDLE (The agent state when not available). 6 - STATE_LOGGED_OUT (The agent state when the agent is not logged in.)
linkDownTi	This is the amount of time, in milliseconds, that it takes before the client is

meUp	alerted that there is a broken connection to the server. The default is 6000 (60 seconds).
wrapUpMode	The agent's Wrap up mode. Values can be: • 0 - No WrapUp • 1 - Automatic Wrap Up • 2 - Manual WrapUp, 3 - Manual WrapUp

The simplest example of getAgentProperties is:

```
<form name="GetPropertyForm"
action="javascript:top.actionFrame.getAgentProperties(window.document.GetPr
opertyForm.property.value)">

<select name="property">

<option value="fullName">fullName</option>

<option value="firstName">firstName</option>

<option value="lastName">lastName</option>

<option value="loginName">loginName</option>

<option value="loginDate">loginDate</option>

<option value="Skill">Skill</option>

<option value="ApplStr">ApplStr</option>

<option value="skillTargetID">skillTargetID</option>

<option value="peripheralID">peripheralID</option>

<option value="loginMediaClass">loginMediaClass</option>

<option value="ACDTerminalID">ACDTerminalID</option>

<option value="isIPTA">isIPTA</option>

<option value="wrapUpEnabled">wrapUpEnabled</option>
```

```
<option value="agentState">agentState</option>

<option value="linkDownTimeUp">linkDownTimeUp</option>

<option value="wrapUpMode">wrapUpMode</option>

</select>

<input type="submit" value="Get Property">

</form>
```

getChatHistory()

This function allows you to retrieve the entire chat history for the current session.

Name	getChatHistory
Description	Returns the entire chat history that is bring stored in the Collaboration Server cache for the current session.
Arguments	none
Usage	getChatHistory()
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • NoSessionEvent - The agent is not currently in session. • ChatMessageEvent - The chat message was posted to the Collaboration Server. This event contains the sessionID, sender name, message history, and a flag that indicates whether the sender is an agent or caller. UnableToGetChatHistory - The chat history could not be received due to an unspecified error.
Requirements	• Agent Must be logged in. Agent must be in session.
Notes	the sendChat() function returns the same event as this function, except the the sendChat() function only returns the latest chat message, whereas this function returns the entire chat history.

The simplest example of getChatHistory is shown below :

```
<form name="getChatHistoryForm"
action="javascript:top.actionFrame.getChatHistory()">


<input type="submit" value="Get Chat History">


</form>
```

getParticipants()

This function allows you to get the participant IDs of the caller and the agent in the current session. The participant IDs are a unique identifier given to each participant in the current session.

Name	getParticipants
Description	Gets the ID's of the Agent and Caller participating in the current session.
Arguments	none
Usage	getParticipants();
Returns	none
Events	• ParticipantListEvent - Returns the agent and caller participant ID for the current Collaboration session. • AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. NoSessionEvent - The agent is not currently in session.
Requirements	• Agent Must be logged in. Agent must be in session.

The simplest example of getParticipants is shown below :

```
<form name="GetParticipantsButton"
action="javascript:top.actionFrame.getParticipants()">


   <input type="submit" value="Get Participants">


</form>
```

login()

Before you can access any of the more useful functions of the API, such as chat or page share, you need to log the agent into the Collaboration Server. Logging in authenticates the user against a username and password in the Collaboration Server database. Once user provides the correct credentials they are connected (or reconnected) to the server.

Name	login
Description	Logs the user into Collaboration Server.
Arguments	• Login name - A String. The agent's login name. Password - A String. The agent's password.
Usage	login(Login Name, Password)
Returns	none
Events	• AgentLoginOkEvent - The user has successfully logged in as an agent. • AgentReconnectOkEvent - The user has successfully reconnected. • InvalidAgentEvent -The user's Login name or Password are invalid. • MaximumAgentsLoggedIn - The maximum licensed number of agents are currently logged in to Collaboration Server has been met an no more agents can log in. GeneralUnexpectedErrorEvent - An unexpected error has occurred and no other information is available.
Requirements	none
Notes	login() is the only API function that you can receive an event for without first calling startEventPolling().

The simplest example of login is shown below:

```
<form name = "LoginForm"
action="javascript:top.actionFrame.login(window.document.LoginForm.username
.value,window.document.LoginForm.password.value)">


<p>Login name: <input type="text" name="username">
```

```
<p>Password: <input type="text" name="password">

<p><input type="submit" value="Login">

</form>
```

Once you have sucessfully logged in or reconnected, you must start event polling by calling the startEventPolling() function from within AgentLoginOkEvent or AgentReconnectOkEvent.

logout()

This functions logs the agent out of the collaboration server.

Name	logout
Description	Logs the user out of Collaboration Server.
Arguments	none
Usage	logout();
Returns	none
Events	• AgentLogoutEvent - The agent has successfully logged out of the Collaboration Server. • AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. UnableToLogoutEvent - The agent cannot log out due to an unspecified error.
Requirements	Agent must be logged in.
Notes	You cannot call stopEventPolling after calling logout().

The simplest example of logout is:

```
<form name="LogoutButton" action="javascript:top.actionFrame.logout()">

<input type="submit" value="Logout">

</form>
```

pageShare()

This function allows you to share pages between the caller and the agent.

Name	pageShare
Description	Shares the page currently displayed in the shared browser
Arguments	Local - Boolean. *true* shares the page currently displayed in the local browser. *false* shares the page currently displayed in the remote browser.
Usage	pageShare(Local)
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • NoSessionEvent - The agent is not currently in session. • FormOrPageShareEvent - Returns the web page URL being shared. UnableToSharePageEvent - The web page could not be shared due to an unspecified error.
Requirements	• Agent Must be logged in. • Agent must be in session. Shared browser must be present.
Notes	You must use bind() before calling this function.

The simplest example of pageShare is shown below :

```
<form name="pushPageForm"
action="javascript:top.actionFrame.pushPage(window.document.pushPageForm.pa
ge.value)">

<input type="text" name="page" value="http://">

<input type="submit" value="Push Page">

</form>
```

pushPage()

This function allows you to send a URL to the caller's browser.

Name	pushPage
Description	Shares the page specified by URL with the other participant in the session.
Arguments	URL - String. A standard URL.
Usage	pushPage(URL)
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • NoSessionEvent - The agent is not currently in session. • FormOrPageShareEvent - Returns the web page URL being shared. UnableToSharePageEvent - The web page could not be shared due to an unspecified error.
Requirements	• Agent Must be logged in. Agent must be in session.
Notes	You must use bind() before calling this function.

The simplest example of pushPage is shown below :

```
<form name="pushPageForm"
action="javascript:top.actionFrame.pushPage(window.document.pushPageForm.pa
ge.value)">

<input type="text" name="page" value="http://">

<input type="submit" value="Push Page">

</form>
```

sendChat()

This function sends chat text from the agent to the caller.

Name	sendChat
Description	Sends a chat message to all users in the Collaboration session.
Arguments	Message - String. The chat message being sent.
Usage	sendChat(Message)
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • NoSessionEvent - The agent is not currently in session. • ChatMessageEvent - The chat message was posted to the Collaboration Server. This event contains the sessionID, sender name, message, and a flag that indicates whether the sender is an agent or caller. UnableToSendChat - The chat message could not be sent due to an unspecified error.
Requirements	• Agent Must be logged in. Agent must be in session.

The simplest example of sendChat is shown below :

```
<form name="sendChatForm"
action="javascript:top.actionFrame.sendChat(window.document.sendChatForm.me
ssage.value)">

<textarea  name="message"></textarea>

<br><input type="submit" value="Send Chat">

</form>
```

setReady()

This function allows you to change the agent's ready state. If the agent's ready state is true then the agent is automatically connected to callers. If the agent's ready state is set to false then the agent will not connect to a caller until his agent state is set to true.

Name	setReady
Description	Sets the agent to be *Ready* or *Not Ready*.
Arguments	Ready - Boolean. *true* requests that agent be set to ready, *false* requests that agent be set to not ready.
Usage	setReady(Ready);
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • AgentReadyEvent - Returns the agent ready state, *true* if the agent is ready, otherwise *false*. • ReadyNotChangeEvent - *AlreadyAvailable* or *AlreadyUnavailable* exception is thrown is you try to call setReady() with the same value twice in a row. UnableToSetReadyEvent - An unknown error occurred while trying to set the Ready state.
Requirements	Agent Must be logged in.

The simplest example of setReady is shown below :

```
<form name="setReadyForm"
action="javascript:top.actionFrame.setReady(window.document.setReadyForm.re
adyState.value)">

<select name="readyState">

<option value="false">false</option>

<option value="true">true</option>
```

```
</select>

<input type="submit" value="Set Ready">

</form>
```

setWrapUp()

This function allows you to enable or disable wrap up for the agent.

Name	setWrapUp
Description	Allows or disallows the agent from entering wrap up mode at the end of a session.
Arguments	Enable Wrap Up - Boolean. *true* allows the agent to enter wrap up at the end of a sessions. *false* prohibits the agent from entering wrap up mode at the end of a session
Usage	setWrapUp(EnableWrapUp);
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • AgentWrapUpEvent - Returns whether or not the agent is enabled or disabled for wrap up. UnableToSetWrapUpEvent - An unknown error occurred while trying to set the wrap up mode.
Requirements	Agent Must be logged in.

The simplest example of setWrapUp is shown below :

```
<form name="setWrapUpForm"
action="javascript:top.actionFrame.setWrapUp(window.document.setWrapUpForm.
wrapUp.value)">


<select name="wrapUp">


<option value="false">false</option>


<option value="true">true</option>


</select>
```

```
<input type="submit" value="Set Wrap Up">

</form>
```

startEventPolling()

This function is used to start polling for events. You must start this function after a successful login or reconnect.

Name	startEventPolling
Description	Activates the polling applet to get events for a user.
Arguments	none
Usage	startEventPolling()
Returns	none
Events	none
Requirements	none

The simplest example of startEventPolling is shown below:

Note: This function should be called from within your cssHandler.js file, after an AgentLoginOkEvent or AgentReconnectOkEvent.

```
function AgentLoginOkEvent()

{

alert('Agent Connect Ok!');

startEventPolling();

}
```

startWrapUp()

This function notifies the Collaboration Server that the agent is starting wrap up.

Name	startWrapUp
Description	Notifies the Collaboration Server that the agent has started Wrap Up.
Arguments	none
Usage	startWrapUp()
Returns	none
Events	• AgentNotLoggedInEvent - The agent was not logged into the Collaboration Server. • NoSessionEvent - The agent is not currently in session. • WrapUpStartEvent - Confirmation that Wrap Up for this agent has been started. • WrapUpUnavailable - Wrap Up for this agent is not available. UnableToStartWrapUp - Unable to start Wrap Up due to an unspecified error.
Requirements	• Agent Must be logged in. Agent must be in session.
Notes	You should only call this function *after* you receive a WrapUpNotifyEvent, which is automatically generated by the disconnect() function if the agent has wrapUp enabled.

The simplest example of startWrapUp is shown below :

```
<form name="startWrapUpFrom"
action="javascript:top.actionFrame.startWrapUp()">

<input type="submit" value="Start Wrap Up">

</form>
```

stopEventPolling()

This function is included for completeness. If you call stopEventPolling() in your application then events are no longer received by the client. If you wish to log out the agent you should use logout().

Polling is done by an applet loaded in API.jhtml and is not connected to the logout() function. Polling may stop if the page containg the polling applet is unloaded.

Name	stopEventPolling
Description	Deactivates the polling applet to get events for a user.
Arguments	none
Usage	stopEventPolling()
Returns	none
Events	none
Requirements	none

The simplest example of stopEventPolling is shown below :

```
<form name="stopPolling"
action="javascript:top.actionFrame.stopEventPolling()">

<input type="submit" value="Stop Event Polling">

</form>
```

Events

AddRequestElapsedEvent

This event occurs when a new caller enters the agent's queue and the agent was not logged in when the call entered the queue. This event is identical to AddRequestEvent except that it has an addition argument, elapsedTime.

Arguments:

- requestID - Integer. A unique ID for the session request.
- requestName - String. The full name of the caller.
- requestSkill - String. The name of the skill that is being requested by the caller.
- elapsedTime - Time. The amount of time that the caller has been waiting in the queue before the agent logged in.

Example:

```
function AddRequestElapsedEvent(requestID, requestName,
requestSkill,elapsedTime)


{


alert('Caller Added to The Queue. Name: ' + requestName + ' Skill: ' +
requestSkill + ' Wait Time: ' + elapsedTime);


}
```

AddRequestEvent

This event occurs when a new caller enters the agent's queue and the agent is currently logged in.

Arguments:

- requestID - Integer. A unique ID for the session request.
- requestName - String. The full name of the caller.
- requestSkill - String. The name of the skill that is being requested by the caller.

Example:

```
function AddRequestEvent(requestID, requestName, requestSkill)

{

alert('Caller Added to The Queue. Name: ' + requestName + ' Skill: ' +
requestSkill);

}
```

AddSessionEvent

This event occurs after a caller join a session with the agent.

Arguments:

- sessionID - Integer. A unique ID for the session.
- agentParticipantID - Integer. A unique ID for the agent participating in this session.
- callerParticipantID - Integer. A unique ID for the caller participating in this session.
- callerParticipantName - Integer. The full name of the caller.
- requestID - Integer. A unique ID for the session request.

Example:

```
function AddSessionEvent(sessionID,agentParticipantID,callerParticipantID,
callerParticipantName,requestID)

{

alert('New Session with ' + callerParticipantName + ' has begun.');

}
```

AgentAlreadyLoggedIn

This event is returned if you attempt to login() in to the Collaboration Server, but the agent is already logged in.

This event has no arguments..

Example:

```
function AgentAlreadyLoggedIn()

{

alert('You are already logged in!');

}
```

AgentLoginOkEvent

This event is returned after a successful login() in to the Collaboration Server. You should startEventPolling() as soon as you receive this event, so that any other events can be caught by your application.

This event has no arguments.

Example:

```
function AgentLoginOkEvent()

{

alert('Agent Connect Ok!');

startEventPolling();

}
```

AgentLogoutEvent

This event occurs after a successful logout().

This event has no arguments.

Example:

```
function AgentLogoutEvent()

{

alert('Agent Logged Out!');

}
```

AgentNotLoggedInEvent

This event occurs if the agent tries to access any of the functionality of the API without first having logged in.

This event has no arguments.

Example:

```
function AgentNotLoggedInEvent()

{

alert('Agent Not Logged In!');

}
```

AgentPropertyEvent

This event occurs after a successful call to getAgentProperties(). The event returns the *key* and *value* of the requested property (key).

Arguments:

- key - String. The name of the property requested.
- value - String. The value of the property requested.

Example:

```
function AgentPropertyEvent(key,value)

{

alert('The value for ' + key + ' is "' + value + '".');

}
```

AgentReadyEvent

This event occurs after a successful call to setReady().

Argument: ready - Boolean. *true* indicates that the agent state is set to ready (1). *false* indicates that the agent state has ben set to idle (2). Callers are pushed to agents only if they are set to ready (1).

Example:

```
function AgentReadyEvent(ready)

{

alert('Agent ready state set to "' + ready + '".');

}
```

AgentReconnectOkEvent

This event is returned when a user has successfully re-connected to the Collaboration Server using login(). You should startEventPolling() as soon as you receive this event, so that any other events can be caught by your application.

Likely events that are received immediately after this event is received include:

- AddRequestEvent - Occurs if a caller enters the agent's queue immediately after the agent reconnects.
- AddRequestElapsedEvent - Occurs if a caller has been waiting in the agent's queue while the agent was disconnected.
- AddSessionEvent - Occurs if the agent is set to ready when he is reconnected and a caller is waiting in queue or was previously in session with the agent.
- ChatMessageEvent - Occurs if the agent was in session with a caller before disconnecting. It may be useful to use getChatHistroy() to recreate the entire chat that was lost when the agent disconnected.
- FormOrPageShareEvent - Occurs if a caller pushed or shared a page or shared a form while the agent was disconnect (or right after the agent logged in).

You should also use getAgentProperties() to determine the agent state, since it will be in the same state as when the agent disconnected.

This event has no arguments.

Example:

```
function AgentReconnectOkEvent()

{

alert('Agent Reconnected!');

startEventPolling();

}
```

AgentWrapUpEvent

This event occurs after a successful call to setWrapUp().

Argument: wrapUpEnable - Boolean. *true* indicates that Wrap Up has been enabled. *false* indicates that Wrap Up has been disabled.

Example:

```
function AgentWrapUpEvent(wrapUpEnable)

{

alert('Wrap Up set to "' + wrapUpEnable + '".');

}
```

ChatMessageEvent

This event occurs after a successful call to sendChat() or getChatHistory().

When called from sendChat() (or if chat is sent by the caller) this event occurs once. When called from getChatHistory() this event occurs as many times as there are chat messages for this session.

Arguments:

- sessionID - Integer. The unique for the current session.
- senderName - String. The full name of the sender of the message.
- messageText - String. The text of the chat message.
- isAgent - Boolean. *true* indicates that the sender is the agent. *false* indicates that the sender is the caller.

Example:

```
function ChatMessageEvent(sessionID, senderName, messageText, isAgent)

{

if(isAgent == true)

{

alert('Message Sent to caller: ' + messageText);

}

else if(isAgent == false)

{

alert('Message from ' + senderName = ' is: ' + messageText);

}

}
```

DropRequestEvent

This event occurs when a caller leaves the agent's skill queue, either because an agent is entering a session with he caller, or the caller has disconnected.

Arguments:

- requestID - Integer. The unique ID for the request.
- requestName - The full name of the caller.

Example:

```
function DropRequestEvent(requestID,requestName)

{

alert('DropRequestEvent requestName = ' + requestName + ' AND requestID = '
+ requestID);

}
```

DropSessionEvent

This event occurs after a disconnect() or after the caller disconnects from a session.

Arguments:

- sessionID - Integer. The uniqie ID for the session.
- callerParticipantName - String. The full name of the caller.

Example:

```
function DropSessionEvent(sessionID, callerParticipantName)

{

alert('Session with ' + callerParticipantName + ' has ended.');

}
```

FormOrPageShareEvent

This event occurs after a successful formShare() or pageShare().

Arguments:

- url - String. The URL that is being shared.
- agentIsSender - Boolean. *true* indicates that the agent was the sender of the page or form. *false* indicates that the caller was the sender.

Example:

```
function FormOrPageSharedEvent(url, agentIsSender)

{

if(agentIsSender == true)

{

alert('The URL "' + url + '" has been sent to the caller.');

}

else if(agentIsSender == false)

{

alert('The URL "' + url + '" has been received from the caller.');

}

}
```

GeneralUnexpectedErrorEvent

This event is received if there is an unexpected error.

This event has no arguments.

Example:

```
function GeneralUnexceptedErrorEvent()

{

alert('General Unexcepted Error!');

}
```

InvalidAgentEvent

This event occurs if the user has not provided the correct Login name or Password while attempting to login().

Arguments: `invalidUsername` - Boolean. *true* indicates that the Login name is invalid. *false* indicates that the password is invalid for the username that was provided.

Example:

```
function InvalidAgentEvent(invalidUsername)

{

if(invalidUsername)

alert('Invalid Username!')

else

alert('Wrong Password!');

}
```

InvalidAgentProperty

This event is returned if you attempt to call getAgentProperties() with an invalid property.

Argument: key - String. The name of the property requested.

Example:

```
function InvalidAgentPropertyEvent(key)

{

alert('Error getting property "' + key + '"!');

}
```

LinkDown

This event occurs if the agent's client loses its connection to the Collaboration Server.

This event has no arguments.

Example:

```
function linkDown()

{

alert('You have lost connection to the Collaboration Server');

}
```

MaximumAgentsLoggedIn

This event occurs if an agent tries to login() while the maximum number of agents are already logged in to the Collaboration server.

This event has no arguments.

Example:

```
function MaximumAgentsLoggedIn()

{

alert('Maximum Number of Agents are Logged-in! Login Denied.');

}
```

NoSessionEvent

This event occurs if you try to call a session-based function (such as getParticipants() or getChatHistory()) and the agent is not currently in a session.

This event has no arguments.

Example:

```
function NoSessionEvent()

{

alert('You are not in session! You must be in session to be able to use
this feature.');

}
```

UnableToLogoutEvent

This event occurs if an agent tries to logout(), but the logout fails due to an unspecified error.

This event has no arguments.

Example

```
function UnableToLogoutEvent()

{

alert('Unable to Log Out!');

}
```

ParticipantListEvent

This event occurs after a successful call to getParticipants(). This event returns the participant IDs for the agent and the caller.

Arguments:

- agentParticipantID - Integer. The unique ID of the agent for this session
- callerParticipantID - Integer. The unique ID of the caller for this session

Example:

```
function ParticipantListEvent(agentParticipantID,callerParticipantID)

{

alert('Agent's Participant ID is: "' + agentParticipantID + '".\n The
Caller's Participant  ID is"' + callerParticipantID + '".');

}
```

ReadyNotChangeEvent

This event occurs if you make a call to setReady() to change the state, but the agent ready state is already in the state that you have requested.

Argument: ready - Boolean. *true* indicates that the agent state is set to ready (1). *false* indicates that the agent state has ben set to idle (2). Calls are pushed to agents only if they are set to ready (1).

Example:

```
function ReadyNotChangeEvent(ready)

{

alert('Agent ready is already "' + ready + '". No Change.');

}
```

UnableToCancelWrapUpEvent

This event occurs after an unsuccessful call to cancelWrapUp() due to an unspecified error.

This event has no arguments.

Example:

```
function UnableToCancelWrapUpEvent()

{

alert('UNABLE To CANCEL wrap up!');

}
```

UnableToDisconnectEvent

This event occurs if the agent is unable to disconnect() from a session.

This event has no arguments.

Example:

```
function UnableToDisconnectEvent()

{

alert('Unable to Disconnect from session');

}
```

UnableToEndWrapUpEvent

This event occurs after an unsuccessful call to endWrapUp() due to an unspecified error.

This event has no arguments.

Example:

```
function UnableToEndWrapUpEvent()

{

alert('UNABLE To END wrap up!');

}
```

UnableToGetChatHistoryEvent

This event occurs after an unsuccessful call to getChatHistory() due to an unspecified error.

This event has no arguments.

Example:

```
function UnableToGetChatHistoryEvent()

{

alert('UNABLE To Get Chat History!');

}
```

UnableToSendChatEvent

This event occurs after an unsuccessful call to sendChat() due to an unspecified error.

This event has no arguments.

Example:

```
function UnableToSendChat()

{

alert('Could Not Send Chat!');

}
```

UnableToSetReadyEvent

This event occurs if you make a call to setReady(), but the state was not changed due to an unspecified error.

This event has no arguments.

Example:

```
function UnableToSetReadyEvent()

{

alert('Unable To Set Agent Ready State!');

}
```

UnableToSetWrapUpEvent

This event occurs after an unsuccessful call to setWrapUp(). The agent Wrap Up mode is not changed to an unspecified or unknown error.

This event has no arguments.

Example:

```
function UnableToSetWrapUpEvent()

{

alert('Error Setting Wrap Up!');

}
```

UnableToSharePageEvent

This event occurs after an unsuccessful call to formShare() or pageShare() due to an unspecified error.

This event has no arguments.

Example:

```
function UnableToSharePageEvent()

{

alert('Could Not Share Page!');

}
```

UnableToStartWrapUpEvent

This event occurs after an unsuccessful call to endWrapUp() due to an unspecified error.

This event has no arguments.

Example:

```
function UnableToStartWrapUpEvent()

{

alert('UNABLE To START wrap up!');

}
```

UserDisconnectEvent

This event occurs after a caller disconnects from a session. A wrapUpNotifyEvent is also generated after this event if wrap up is enabled - see setWrapUp().

Arguments:

- callerParticipantID - Integer. The unique ID of the caller.
- callerParticipantName - String. The name of the caller.
- requestID - Integer. The Unique ID of the request.

Example:

```
function UserDisconnectEvent(callerParticipantID,
callerParticipantName,requestID)

{

alert('The caller, ' + callerParticipantName + ', has disconnected from the
session.');

}
```

WrapUpNotifyEvent

This event occurs after a disconnect() or if the caller disconnects from the session if Wrap Up is enabled for the agent (either manually or by default).

This event has no arguments.

Example:

```
function WrapUpNotifyEvent()

{

alert('WrapUp Notify Event');

}
```

WrapUpNotStartedEvent

This event occurs after a call to endWrapUp() when Wrap Up is not started (using startWrapUp() or if Wrap Up is started automatically after a disconnect() because of the wrap up setting in apiAgents.properties).

This event has no arguments.

Example:

```
function WrapUpNotStartedEvent()

{

alert("Wrap Up has not been started!');

}
```

WrapUpStartEvent

This event occurs after a successful call to startWarpUp().

This event has no arguments.

Example:

```
function WrapUpStartEvent()

{

alert('Beginning Wrap Up!');

}
```

WrapUpUnavailableEvent

This event occurs after an unsuccessful call to startWrapUp(), endWrapUp() or CancelWrapUp().

This event has no arguments.

Example:

```
function WrapUpUnavailableEvent()

{

alert('WrapUp Unavailable!');

}
```

Event Handler Files

Event Handler Files Introduction

The two event handler files, APIHandler.js and APILiteHandler.js, have been included at the end of this document for your reference. They are shown in the same state as when the are shipped with Collaboration Server. Most of the functions are blank, and must be modified for your specific application.

APIHandler.js

This is the complete set of event handler functions as they are distributed with Cisco Collaboration Server. You must modify each of the functions in this file to work with your application.

```
////////////////////////////////////////////////////////////////////////////
///////////////////////////


///******  API event handlers -Edit this file for your own application
specific implementation    ///


////////////////////////////////////////////////////////////////////////////
///////////////////////////


// This event occur after a sucessful logout().


function AgentLogoutEvent()


{


// application specific implementation go here...


}


// This event occur after calling getParticipants() sucessfully.


// Event contains agent and caller participantId


// Note: For this release, we only support one agent and one caller in the
session. We don't support meetingMe also.


function ParticipantListEvent(agentParticipantID,callerParticipantID)


{


// application specific implementation go here...


}


// This event occur after calling getAgentProperties(key)
// Event contains property with key/value pair.

function AgentPropertyEvent(key,value)

{

// application specific implementation go here...

}

// This event occur after a sucessful call to setReady(true or false).

// Event return agent new ready state. true or false

function AgentReadyEvent(ready)

{

// application specific implementation go here...

}

// This event is removed

// Event return wrap up state(wrapUpMode in apiAgent.properties file) --> 0
= No WrapUp, 1 = Automatic Wrap Up , 2 = Manual WrapUp, 3 = Manual WrapUp

//function AgentWrapUpModeEvent(wrapUpMode)

//{

// application specific implementation go here...

//}

// Event return wrapup status -> setWrapUp(enable) or setWrapUp(disable)

// wrapUpEnable could be true or false

function AgentWrapUpEvent(wrapUpEnable)

{

// application specific implementation go here...
}

// This event will be generated when participant is dropped from session
and wrap up is enable.

// Agent should call startWrapup or cancelWrapup after receiving this
event.

function WrapUpNotifyEvent()

{

// application specific implementation go here...

}

// This event occurs after a sucessful formShare() or pagePage().

// url is the url been shared.

// agentIsSender is true is sender is agent, otherwise, it is false

function FormOrPageSharedEvent(url, agentIsSender)

{

// application specific implementation go here...

if(agentIsSender == true)

{

// If you need to do something after agent do pageShare, here is the place

}

else if(agentIsSender == false)

{

// If you need to do something after caller do pageShare, here is the place

}

}
```

```
// Chat event

// sessionID --> is session unique ID.

// sendName -- > is sender name(could be agent or caller)

// messageText -> messager sent by agent or caller

// isAgent --> flag used to indicate sender is agent or caller

function ChatMessageEvent(sessionID, senderName, messageText, isAgent)

{

// application specific implementation go here...

if(isAgent == true)

{

// Chat sent from agent..

}

// Message from caller

else if(isAgent == false)

{

// Chat sent from caller..

}

}

// This event will occur when caller disconnect from session event.

// participantId --> is caller unique ID and

// participantName --> is caller fullname

// requestRef --> is unique ID caller(request) before they join into
session.
```

```
function UserDisconnectEvent(callerParticipantID,
callerParticipantName,requestID)

{

// application specific implementation go here...

}

// This event occur after sucessful call to startWraupUp().

function WrapUpStartEvent()

{

}

// This event been generated when agent and caller get into session.

function AddSessionEvent(sessionId,agentParticipantID ,callerParticipantID,
callerParticipantName,requestID)

{

// application specific implementation go here...

}

// This event been generated when agent and caller leave the session.

function DropSessionEvent(sessionID, callerParticipantName)

{

// application specific implementation go here...

}

// Agent will receive this event if there is new caller get into the skill
queue this agent belong to.

function AddRequestEvent(requestID, requestName, requestSkill)

{

// application specific implementation go here...
}

// Agent will receive this event during login if there is request waiting
in the skill queue that agent belong to.

// Agent will also receive this event if agent is reconnecting to the
server and there is request in the skill queue.

// elapsedTime is the time(milisecond) caller has been waiting in that
skill queue before agent login.

function AddRequestElapsedEvent(requestID, requestName,
requestSkill,elapsedTime)

{

// application specific implementation go here...

}

// Request dropped event.

function DropRequestEvent(requestID,requestName)

{

// application specific implementation go here...

}

// Remove - use getAgentProperties to get skill

// Skills belong to this agent event - this event genereated when agent
first logging in.

//function AddSkillEvent(skillName)

//{

// application specific implementation go here...

//}

// Agent Login successfully

function AgentLoginOkEvent()
```

```
{

// application specific implementation go here...

}

// Agent reconnect successfully

function AgentReconnectOkEvent()

{

// application specific implementation go here...

}

///////////////////////////////////////////////////////////////////////////
////////

/////////////////////// Error related events
///////////////////////////////////////

///////////////////////////////////////////////////////////////////////////
////////

// Agent is not logged in event.

function AgentNotLoggedInEvent()

{

// application specific implementation

}

// Agent is not in session event.

function NoSessionEvent()

{

// application specific implementation

}

// Invalid property event.
```

```
function InvalidAgentPropertyEvent(key)

{

// application specific implementation

}

// Ready button already in this state.

function ReadyNotChangeEvent(ready)

{

// application specific implementation

}

// Unable to enable or disable wrap up.

function UnableToSetWrapUpEvent()

{

// application specific implementation

}

// Unable to share page event.

function UnableToSharePageEvent()

{

// application specific implementation

}

// Unable to share form event.

function UnableToShareFormEvent()

{

// application specific implementation
}

// Unable to send chat event.

function UnableToSendChatEvent()

{

// application specific implementation

}

// Unable to disconnect event.

function UnableToDisconnectEvent()

{

// application specific implementation

}

// Unable to start wrapup

function UnableToStartWrapUpEvent()

{

}

// Unable to end wrapup

function UnableToEndWrapUpEvent()

{

}

// Unable to Cancel wrap up

function UnableToCancelWrapUpEvent()

{

}
```

```
// Unable to logout agent

function UnableToLogoutEvent()

{

}

//Unable to set ready event

function UnableToSetReadyEvent()

{

}

//WrapUp unavailable

function WrapUpUnavailableEvent()

{



}

// Unable to get chat history

function UnableToGetChatHistoryEvent()

{

}

// Wrapup is not started

function WrapUpNotStartedEvent()

{

}

function linkDown()

{
```

```
alert('linkDown');

}

function InvalidArgumentTypeEvent()

{

alert('InvalidArgumentTypeEvent');

}
```

APILiteHandler.js

This is the "lite" set of handler functions that contain only login events. This should be used in conjunction with APILite.jhtml to login and then open a new window to that includes the complete API.

You must modify each of the functions in this file to work with your application.

```
///////////////////////////////////////////////////////////////////////////
/

// /////////////All agent login related
events//////////////////////////////

///////////////////////////////////////////////////////////////////////////
/



// 1) Agent sucessfully logged in.

function AgentLoginEvent(isReconnect)

{

/// application specific implementation go here...

}

// 2) InvalidUsername == true if username is invalid, otherwise  password
is invalid

function InvalidAgentEvent(invalidUsername)

{

/// application specific implementation go here...

}

// 3) Agent already logged in at other machine.
```

```
function AgentAlreadyLoggedIn()

{

/// application specific implementation go here...

}

// 4) CCS has the maximum number of concurrent agent logged-in

function MaximumAgentsLoggedIn()

{

// application specific implementation go here...

}

// 5) General unexpected login error

function GeneralUnexceptedErrorEvent()

{

// application specific implementation go here...

}
```