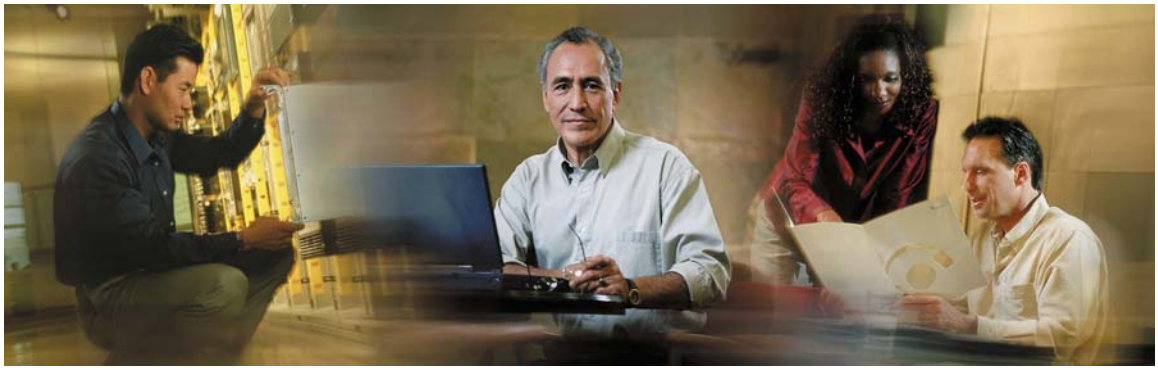


Cisco Collaboration Server Session Platform SDK User's Guide

Version 5.0

Table Of Contents

Introduction

Cisco Collaboration Server Session Platform SDK

Getting Started

The Cisco Collaboration Server Session Platform SDK, or CSSP SDK, is a platform for building collaborative Internet applications. With CSSP SDK, you can turn your Java application into one that can be shared by different people across the Internet. You should read the user guide first. Then, when you are finished, use this document as a guide to begin programming.

This guide shows you how to get started using CSSP SDK. Follow the steps and the procedures listed below to get your collaboration application working. You can experiment with some sample applications first. Then, when you are ready, you can create your own applet.

Things to Remember – how to make the most of CSSP SDK.

Be aware of network communications

Be aware of what other people in the session may be doing.

Send as little information as possible

Send information in chunks.

Feedback to the user.

Try to keep local state, and build events to let you know when it changes.

Test, test, test

Terms

JHTML - A language similar to HTML that generates pages from hybrid HTML/Java files. JHTML is similar to CGI or ASP, and is the predecessor of JSP.

CSSP SDK - Stands for Collaboration Server Session Platform.

Where the files are

Locating these files is the first step to getting started with CSSP SDK. The first three files are necessary for the system to work. The second set of files are sample applications. Review and change these sample applications to get a better comprehension of CSSP SDK's features.

These files are used by the system:

<Cisco_CS>\pub\html\capi: JHTML files for joining a test session and configuring the server.

<Cisco_CS>\pub\applets\Com\WebLine\CAPI: The CAPI library (class files)

<Cisco_CS>\servlet\Com\WebLine\CAPI: The same files as in ...pub/applets...

These files are sample applications. You can alter these applications as you like:

<Cisco_CS>\pub\html\capi\sample

<Cisco_CS>\pub\applets\Com\WebLine\CAPI\Sample

<Cisco_CS>\servlet\Com\WebLine\CAPI\Sample

Sample applications

This is a list of the sample applications available in **<Cisco_CS>\pub\applets\Com\WebLine\CAPI\Sample**. This allows you to experiment with CSSP SDK before creating your own application.

HelloWorld1: The most basic application of CSSP SDK, this application covers basic concepts for getting started and sending events. For more information, see **HelloWorld1: Getting Started, Sending Events** in the *Examples* section of the User's Guide.

HelloWorld2: Slightly more advanced CSSP SDK features than HelloWorld1. This sample covers subscription filters and how to get participant information. For more information, see **HelloWorld2: Using Subscription filters, Getting Participant Information** in the *Examples* section of the User's Guide.

HelloWorld3: This final application involves more advanced CSSP SDK features, such as custom shared data and history. For more information, see **HelloWorld3: Using Custom Shared Data, Using History** in the *Examples* section of the User's Guide.

Magnetic Poetry: This application resembles the popular refrigerator magnets. Magnetic poetry displays a collection of tiles, each with a word or phrase. You reposition the tiles to form a sentence. You can also create or discard any tile you want. For more information, see **Magnetic Poetry** in the *Examples* section of the User's Guide.

Tic Tac Toe: Tic Tac Toe allows two people, using different browsers, to play tic tac toe against each other. Other participants can join in and watch the game as it progresses. For more information, see **Tic Tac Toe: Stateless and Remote Connection Strategies** in the *Examples* section of the User's Guide.

Creating your own CSSP SDK applet

Listed below is the procedure for creating your own CSSP SDK applet. Each of these five major steps is described in detail further down in this document.

Build applet java files and compile them into class files

Build applet jhtml page

How to install your application into CSSP SDK

How to run your CSSP SDK application in test mode

How to run your CSSP SDK application in integrated mode

Build applet java files and compile them into class files

When creating a CSSP SDK application, you need to compile it against the CSSP SDK library. Follow these directions to compile a new application.

Specify where the class files are. Class files are located in **<Cisco_CS>\pub\applets\CAPI**.

Specify the JDK version. At Cisco, we use JDK 1.1, but it is recommend that you use JDK 1.1.8.

Enter this command line to compile a new CAPI application:

%JAVA_HOME%\bin\javac -classpath"%JAVA_HOME%\lib\classes.zip;<Cisco_CS>\pub\applets" <your .java files>

Build applet JHTML page

You need to create a jhtml page to run your application in. You should use **HelloWorld1.jhtml** as an example of how to pass parameters to your applet. To do this, simply change the html to point to your applet.

How to install your application into CSSP SDK

Installing your CSSP SDK application posts it on the server and makes it available for you to use. After this, you can access your CSSP SDK application from the server.

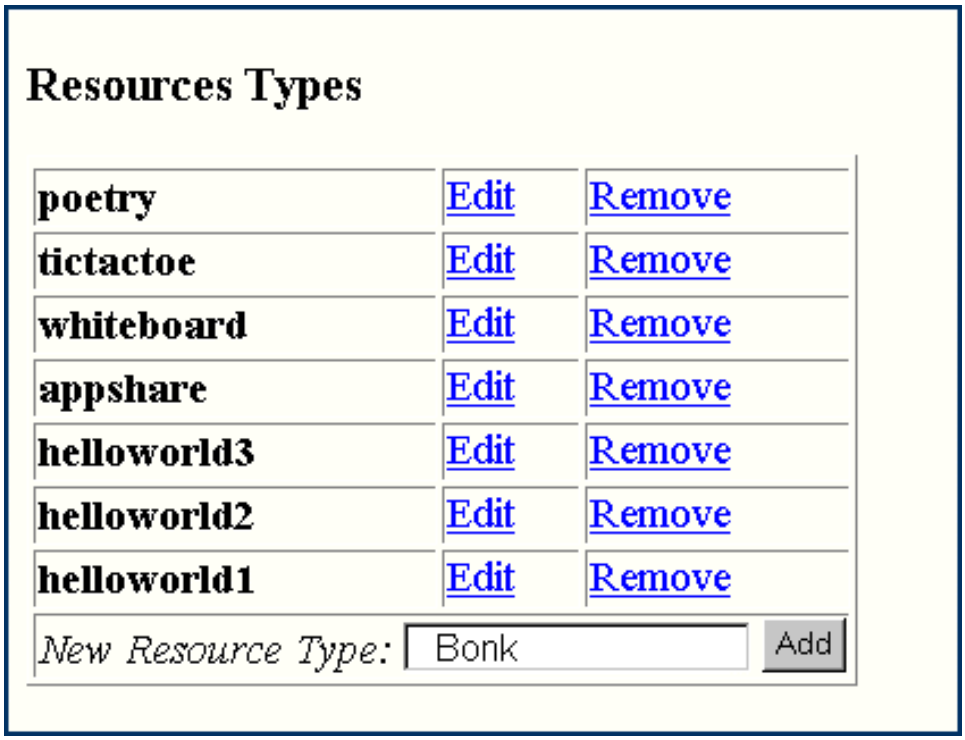
Shut off Collaboration Server Server.

Copy your packages into **<Cisco_CS>\pub\applets and <Cisco_CS>\servlet**.

Copy your page to a free area on the webserver.

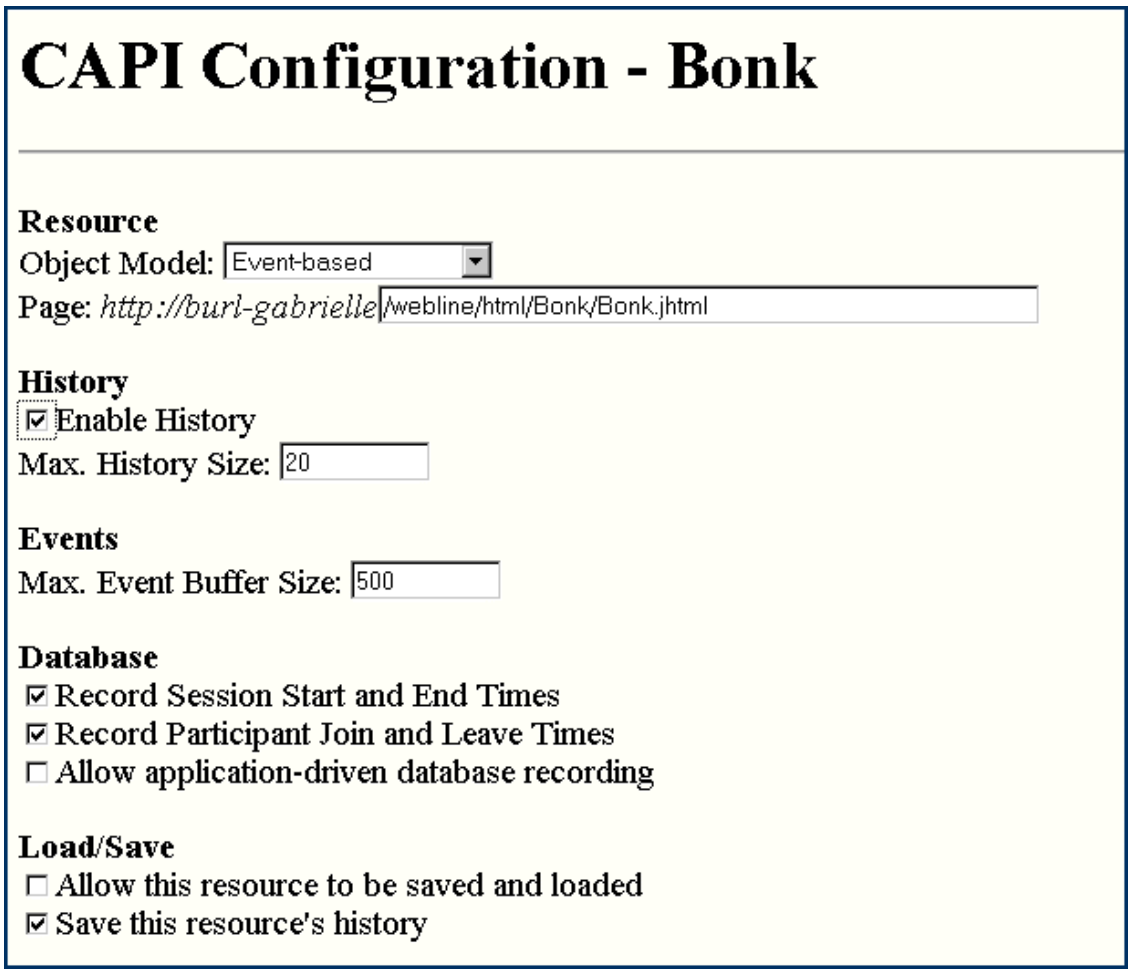
Start Collaboration Server Server.

Load **http://<server>/Cisco_CS/html/capi/capiconfig.jhtml**.



Go to the **Resource Types** area, add your new applet and give it a name (see image above). You are brought to a configuration page for your new applet.

Select the appropriate options. For the **page** option, type in the url of the jhtml page you created (see image below).



Click **OK**. Your CSSP SDK application is installed.

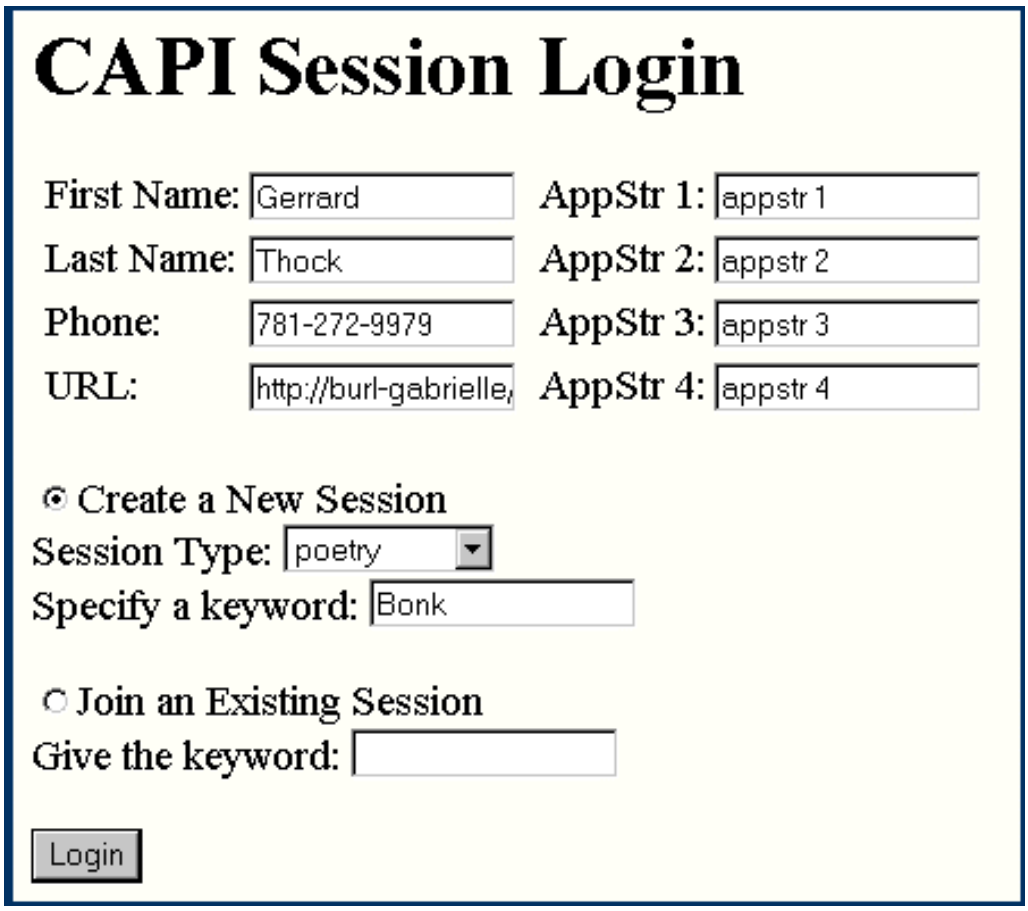
How to run your CSSP SDK application in test mode

Once you've finished creating your application you can launch it from your server and test it.

Load **http://<server>/Cisco_CS/html/capi/capisession.jhtml**.

Enter your user information (see image below). The form has several strings which look like they need to be filled in. If the application doesn't refer to them, you can skip to step 3.

***Note**:* The parameters supplied to the application in the generated HTML depend on those specified in the JHTML page (see How to install a new CSSP SDK application-step 7). If you need additional information, see the Tic Tac Toe sample program. Tic Tac Toe uses an intermediate page to let you choose the type of connection.



Click **create a New Session**.

Select your new application type and specify a keyword.

Click **Login**.

Using another browser (either on the same or a different machine), go to **http://<server>/Cisco_CS/html/capi/capisession.jhtml**.

Click **Join an Existing Session**, and enter the keyword from step 2. Once the session has been created you can access it using the keyword. The session type is used to create the session, and the keyword is then associated with it. The keyword can be any string.

Note: With this option you can easily create multiple sessions of the same type, but it's also easy to forget which keyword goes with which session. We recommend that you avoid punctuation and other special characters when naming keywords, to limit confusion.

Click **login**. You are now in a collaborative session.

How to run your CSSP SDK application in integrated mode

Once your application works, you can launch it from the agent desktop through a link in your script that executes the following javascript:

top.control.document.SRep.launchCAPI (name, type)

where name is an arbitrary name that you decide and type is the resource type that you added during the configuration for your applet.

Preface

Purpose of document

This document serves as a road map for building collaborative applications using Cisco's Collaboration Server Session Platform SDK and a description of Collaboration Server Session Platform SDK's implementation on the Cisco Collaboration Server. It is intended to be an in-depth resource for all the information needed by a collaborative application developer.

Click here for the CSSP SDK Java document.

Naming Change From Previous Release

The Collaboration Server Session Platform SDK is referred to as CSSP SDK in this document. Formerly (pre-version 40. for Collaboration Server), the name was Collaboration API, or CAPI. The name was changed to accurately represent the functionality of the product. Any files referred to as *CAPI* in the document still apply; their names have not changed.

Updates in Session Platform SDK for Version 5.0

Some minor updates were added to the 5.0 release. The following new features were added:

Server-side Logging for the Client-side CAPI Adapter

New methods were added for server-side logging for the client-side CAPI adapter. These methods can be called from inside a CAPI application. The methods pass their arguments backs to the server which then calls the standard logging mechanism.

The following public methods were added:

- `public void log(int verboseLevel, String objName, String message)`
- `public void log(int verboseLevel, String message)`
- `public void log(String objName, String message)`
- `public void log(String message)`

Subscription Filters are Now Able to be Dynamically Modified

There is a new method in this release of the Session Platform SDK that allows you to update subscription filters dynamically. Previously, you were only able to modify subscription filters at the time they were created. The new method is:

- `updateSubscriptionFilters(SubscriptionFilter filter)`

Adapter Files That Include The Above Changes

The following adapter files incorporate the above changes:

- Adapter.java
- IAdapter.java
- BaseAdapter.java
- server\CAPIJHTMLAdapter.java

- server\CAPILocalAdapter.java
- server\CAPIServletAdapter.java

isCollaborating() Method added to BaseAdapter.java

Finally, a method was added to BaseAdapter.java that determines if an application is in collaboration mode:

- `isCollaborating()`

Returns `true` if the application is collaborating, otherwise it returns `false`.

Complete Details Available in Collaboration Server Session Platform SDK JavaDoc

The JavaDoc documentation includes the latest updates to the SDK. You can view the Collaboration Server Session Platform SDK JavaDoc from your web browser at http://<COLLABORATION SERVER>/doc/books/coll_capi/capi_javadoc/index.html, where <COLLABORATION SERVER> is the name or IP address of the server running Cisco Collaboration Server.

Overview

Why Use Cisco Collaboration Server Session Platform SDK

This section describes why you would use the Collaboration Server Session Platform SDK.

Click to read the following sections:

Build collaborative applications

Firewall-friendly communication

Selective concurrence

Assign roles to users of a collaborative application

Build collaborative applications

The Cisco Collaboration Server Session Platform SDK is a platform for building collaborative internet applications, letting different people in different parts of the world interact with each other through a shared application interface. Currently, the accepted model of collaboration over the Internet lets several users interact together on one application, as if all the users were staring over each other's shoulders to view the same computer screen. They take an existing, single-channel application and extend that interface to several participants in a collaborative session. Each participant then has an identical view into that application and can operate on it accordingly. Technologies such as sharing web pages, sharing web forms and remote application sharing all fall in this category.

Sometimes, however, the need arises to do more than just duplicate an existing channel. CSSP SDK provides the tools to make an application aware of its collaborative nature, thus allowing it to exercise more control over what is shared, how it is shared, and who is involved. Individual instances of a collaborative application retain their ability to interact with the operating system and the desktop, while still allowing information to be shared with other participants.

Firewall-friendly communication

Using Collaboration Server Session Platform SDK, applications can implement real-time interactivity between the participants in a session, no matter what kind of network the application sits on. CSSP SDK is aware of firewalls and proxy servers, and can choose the fastest available connection to the Cisco Collaboration Server, whether it is through a direct two-way socket connection, or a slower, firewall-friendly HTTP connection.

Selective concurrence

In other forms of collaboration, an application has no control over what information should be distributed and what information should be kept private. Collaboration Server Session Platform SDK exposes an application to the external coordination that enables collaboration, and allows an application to define its own shared data. Each application can then choose when and what is shared with other participants in the session, using whatever mechanism is appropriate. For example, a simple application will choose to share everything it knows about, while a more complex application may actually prompt the user for permission before sharing sensitive information.

Assign roles to users of a collaborative application

In the current paradigms of collaboration, each user is treated equally. Collaboration Server Session Platform SDK allows a particular application to share the same data between users of different roles. One user may be in a presenter seat, while another may be merely a viewer.

What is Cisco Collaboration Server Session Platform SDK?

This section describes what Cisco Collaboration Server Session Platform SDK is.

Click to read the following sections:

Simple API framework

A robust stand-alone server product

Simple API framework

Collaboration Server Session Platform SDK is a simple, straightforward API framework for building real-time collaborative applets/applications over the Internet. It provides all the tools for creating a multi-participant session, attaching a shared application to it, packaging up that application's state, and distributing it to the other participants in the session. The API portion of the product contains all the objects and methods needed to share the application. The Cisco Collaboration Server provides the back-end for managing the application sharing strategies.

A robust stand-alone server product

Collaboration Server Session Platform SDK can be run in stand-alone development mode, where it supports the creation of sessions and participants based on keywords. A session is created by specifying a keyword, and any person knowing that keyword can join the session. This mode allows a simple, out-of-the-box development environment for CSSP SDK, letting developers get running quickly, without much hassle.

Features Supported by Cisco Collaboration Server Session Platform SDK

This section describes the features supported by Collaboration Server Session Platform SDK.

Click to read the following sections:

Communication Strategies

Event distribution

Application data model

Leadership

Persistence

Reporting

Communication Strategies

Throughout the World Wide Web, applications live in many different places. Some applications are downloaded to the local computer within the confines of a web page, while others are based in a server with an HTML user interface. Some live behind a firewall, while others have more direct access to machines all across the Internet. Collaboration Server Session Platform SDK supports many of these environments, exploiting the advantages of each wherever appropriate.

An application may choose its communications mechanism explicitly, or it may ask CSSP SDK to find the best one for the existing environment. There are currently four different independent strategies for communicating with a Cisco Collaboration Server.

Remote HTTP CSSP SDK

This strategy allows firewall-friendly collaboration over the Internet. Although it is the lowest performing strategy, it is the only one that will work in the general Internet case, complying with both firewall and proxy server standards.

Remote Socket CSSP SDK

This strategy is considerably faster and more responsive than the Remote HTTP strategy, but it will not work in the presence of firewalls. This connection is the most desirable in an Intranet setting, where the network is tightly controlled.

Embedded CSSP SDK

This strategy allows a CSSP SDK client to be run inside the Java Virtual Machine that is running the Cisco Collaboration Server. Applications integrated with Embedded CSSP SDK can integrate with server-side database systems for larger, enterprise class applications.

Stateless CSSP SDK

This strategy is similar to Embedded CSSP SDK but it does not need a running application associated with it. Similar to CGI scripting, this strategy uses JHTML or JSP technology to create a traditional appearing, web-based collaborative application.

Event distribution

Basic to sharing an application in real time is notifying other participants when something happens. Collaboration Server Session Platform SDK allows each application to define its own set of events, and provides the mechanism to both distribute and handle those events. Each event encapsulates a description of what happened in an application, which application it happened in, when the event occurred, as well as some application-defined information about the event.

Since different roles using a shared application may care about different events, CSSP SDK also exposes a subscription model to filter out all events that go unused. Proper use of a subscription model may greatly improve the performance of the shared application.

Application data model

Collaboration Server Session Platform SDK works by packaging up an application's state into a generic format understood by the Cisco Collaboration Server, and then distributing this state to the other participants in the session. The server will maintain this state so that at any point in the session, participants just entering can synchronize their display to the single shared state.

An application defines its own data packets, but different types of applications will have different needs as far as manipulating these packets. Some simple applications simply want to send the packets to the other participants, whereas more complex applications may actually want the server to maintain a collection of these packets. CSSP SDK offers four object models to help. A relatively straightforward application can usually describe its data using one of the following four object models.

Event based Object Model

Events are distributed to all participants in the session. The application does not really maintain any modifiable state, but it does respond to actions. This is ideal for an application that simply responds to user's commands.

Single object based Object Model

A simple object is maintained on the server containing some application-defined state information. This object can be modified throughout the session by each application, and so a record is kept of how the state changes. This is ideal for an application that has a small but consistent amount of data. Additionally, this object model supports the event based object model functionality.

Revision based Object Model

A revision history is kept on the server, including a base-level state description and a series of revisions to it. When the base-level state changes, the revisions are cleared. This is ideal for an application that maintains a rather large description of its state that changes incrementally. For example, a word processing document can be quite large, but usually only single words, sentences, or paragraphs are being edited at any time. A revision-based model will only update the pieces of the state that have changed, and not the entire description. Additionally, this object model supports the event based object model functionality.

Collection based Object Model

A collection of objects is maintained on the server, each one containing a portion of the application's data. These objects can be created, manipulated individually, and deleted, and records are kept of all the modifications. This is ideal for an application that maintains a changing list of independently controlled objects. Additionally, this object model supports the event based object model functionality.

Leadership

Often in a collaborative session, it is desirable for one of the participants to have exclusive control over what is happening in the session, or leadership. Collaboration Server Session Platform SDK supports a built-in means to establish a session leader and prevent other participants in the session from interfering with a presentation at the wrong time.

Persistence

Collaboration Server Session Platform SDK provides a means to save the current state of an application and load it up again into a different session.

Reporting

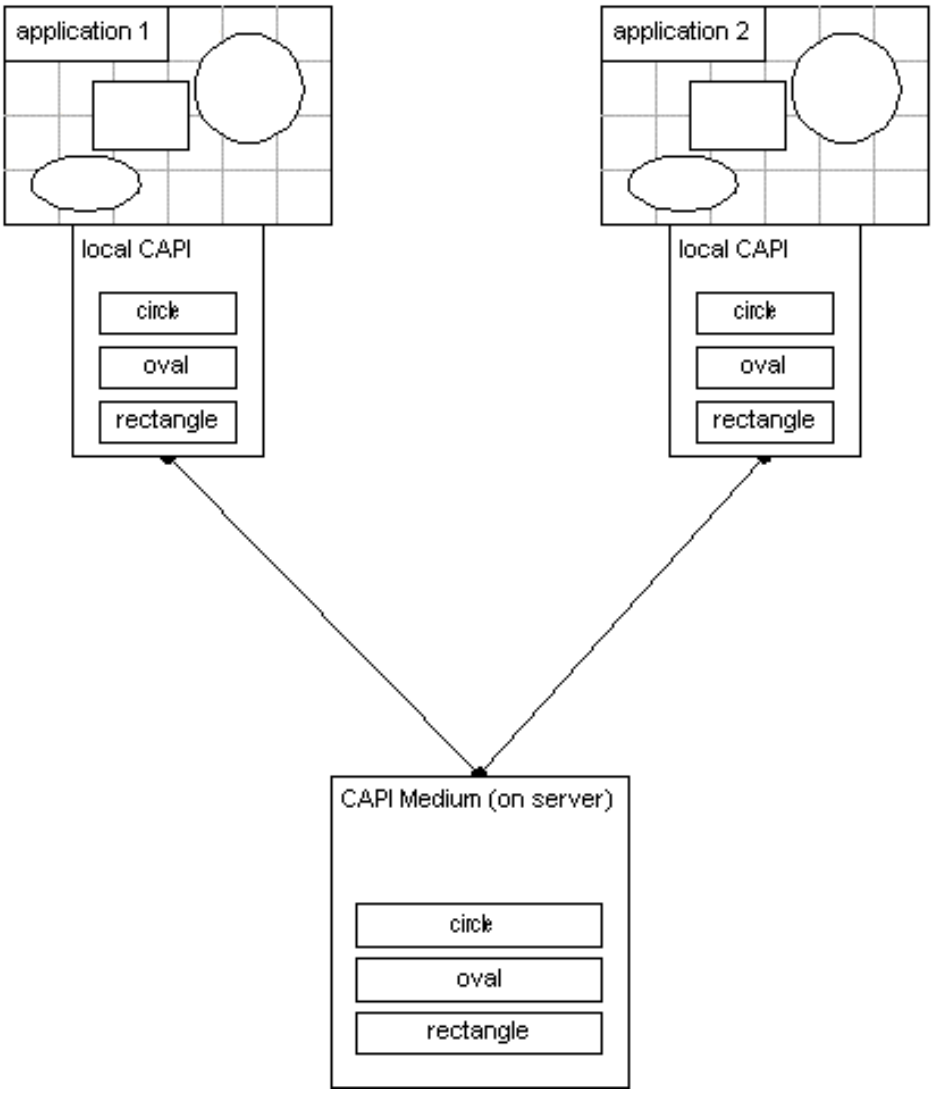
Collaboration Server Session Platform SDK provides hooks to record application-specific events into the Collaboration Server database. An application defines its own set of properties to store.

Programming Semantics

Client-Server Model

Cisco Collaboration Server Session Platform SDK is based on a client-server model, where applications acting as clients, establish identity on the server to participate in a session. Once identity is validated, the clients are allowed access to view and modify the shared data stored in the server's session.

Each CSSP SDK session stored on the Cisco Collaboration Server is responsible for maintaining the distributed state of its associated application, while the actual application running on a participant's desktop maintains a local copy of the application's state. Although an application can decide how closely it chooses to match its local state to the distributed state, typically applications will try to keep these models as synchronized as possible.



When a user interacts with a local application and causes a change in the application's local state, the application is responsible for deciding if that change should affect the distributed state - that is, if the change should be seen by the other participants. When it is decided that the distributed state must be updated, the application packages the change and notifies the server. After the server updates the session's distributed state, it notifies all the participants in the session. It is important to note that the only way the session's distributed state will change is if it is instructed to do so by a client. The server contains no knowledge of how to manipulate the application-specific data, and thus all that logic is left to the client.

Once the server commits the distributed state changes, each local instance of the application receives a notification describing the change. The client updates its local state and manipulates the application's display. At this point, all participants in the session are again synchronized with each other.

Overview of Objects

Although the implementation of Collaboration Server Session Platform SDK relies heavily on its client-server model, the actual programming interface is designed to hide many of these details. The developer must be aware of differences between the client and the server, but CSSP SDK conceals the programmatic separation of these two components. A collaborative application is developed using CSSP SDK in such a way that you need only think of the distributed application as a whole, and not of its client and server portions specifically.

To accomplish this objective, CSSP SDK separates the application's self-defined shared data from its actual sharing with other participants. In doing so, this shared data object can be thought of as the single collaborative component of the application, and it is used in both the client and the server to fulfill all the needs of the shared application end-to-end. The shared data object, or ResourceData class, is extended by the application to meet the application's specific collaborative requirements. The client and the server know how to use the pieces of that object that are appropriate to each side to complete the distributed application.

The following are descriptions of the most important objects in the CSSP SDK programming model. For a more complete list of objects and methods, see the Collaboration Application Programming Interface section of this document.

Click to read the following sections:

ResourceData

Resource Classes

ResourceEvent and IEventHandler

Adapters

History

Database records

ResourceData

This is the shared data object that an application developer defines to interact with the collaborative application, the Collaboration Server Session Platform SDK client, and the Cisco Collaboration Server. The application developer extends this object to

contain state data, perhaps adding accessor methods and helper functions where appropriate. The developer must also implement serialization functions, used by the client and the server to transfer the object back and forth. Finally, this object contains overridable functions that provide loading and saving, and database event recording.

All the interaction that takes place between an application and its ResourceData objects occurs locally. These objects are simply passed around from client to server, but they are not distributed themselves. In fact, they are the targets of the actual distributed objects, known as Resource Classes.

Resource Classes

The family of Resource Classes is the set of distributed objects in the Collaboration Server Session Platform SDK system. These objects all communicate directly with the Cisco Collaboration Server to update the distributed state within a session. There are four Resource Classes, implementing the four object models discussed above. Once an application attaches itself to a session on the server, it uses the session's associated Resource Class as its window into that session.

Resource Classes use ResourceData objects as currency to transfer state between clients and the server. When an application detects a change and must notify the server to update its distributed state, it describes the change using a ResourceData object it defines. It then sends the change to the server through its session's Resource Class object.

The four Resource Class objects that exist are:

EventResource (implements the event based object model)

StateResource (implements the single-object based object model)

DiffResource (implements the revision based object model)

VectorResource (implements the collection based object model)

These objects are all derived from a single BaseResource class that contains the functionality common to all resources; including leadership, history, and participant and session information.

Command Batching

As methods are called on the Resource Class object, it is important to remember that these objects are distributed, even though that fact is largely hidden from the

developer. Resource classes provide a way to batch commands together so they are sent to the server all at once. The other participants consequentially receive any events caused by those batched commands all at once. Batches of commands are also guaranteed to be executed on the server as a synchronous block, preventing any other commands from executing until the batch is complete. Using batches wisely can greatly improve the performance and the usability of a collaborative application.

ResourceEvent and IEventHandler

When a session on the server updates its distributed state, it must then notify all the participants in the session of the change. It does this by constructing a ResourceEvent object, which describes the change, and sending that object to all the participants in the session. This event object describes what happened conceptually, the time that the event happened, which participant caused it, and it carries with it some additional event-specific data. For example, when a ResourceData object is modified, the event will contain the new state of that object. Similarly, when a participant joins the session, the event will contain data about that new participant.

Several types of actions on the server cause events, but most commonly a participant in the session will trigger an event when modifying the session's distributed state. When an application does this, it passes along an event type with the command. This event type is an application-defined value that indicates conceptually how the state changed. This value is carried with the event when it is delivered to the other applications, which may use the value to determine how their local state should now be modified.

Applications handle events by implementing the IEventHandler interface. This interface contains methods for dealing with events that are sent by the server. When an application registers itself with the session, it passes in the object that implements this interface. The CSSP SDK client's communications subsystems will use this object whenever the server has new events queued up for the application.

If there is an error in network communications, or if the application exits unexpectedly, the server will begin to queue up events that a participant does not retrieve. Since the server puts a limit on the size of each participant's queue, the server may purge the events for a participant if they do not reconnect in a short enough time. When this happens, upon reconnecting the application will receive an error indicating that events have been purged. The application should explicitly rebuild its local state, rather than relying on the event stream.

Adapters

Adapters implement the communications strategies described above. In doing so, they hide all the details about socket protocols, HTTP communication, firewall-friendly communications, network errors, etc., from the application. An application

must create an adapter to communicate, but once this happens there is little interaction between the adapter and the application.

The four adapters that exist are:

HttpAdapter *(Remote HTTP Connection Strategy)*

SocketAdapter *(Remote Socket Connection Strategy)*

LocalAdapter *(Embedded Collaboration Server Session Platform SDK Connection Strategy)*

JHTMLAdapter *(Stateless Collaboration Server Session Platform SDK Connection Strategy)*

History

Applications have access to the history of a Resource Class's changes during a session. This can be thought of as a kind of replay feature, where an application can request the distributed state at a certain point in the session, and then enumerate through the events that followed until the present time.

The server archives state information and events to provide this information, but limits are imposed on how far back the history will reach. A system administrator may configure these limits to take advantage of the server's computing and storage resources. They exist to prevent the server from sacrificing real-time interaction for the sake of archived history information. The history of a session should be used sparingly, and all information that is real-time and interactive should be designed into the object model.

Database records

The Cisco Collaboration Server maintains a database for recording events that occur in the system. A collaborative application has access to a portion of this database for recording significant events that occur in it. When an application updates its session's distributed state, the server asks the ResourceData object if, based on the event type, it would like to record itself into the database. The ResourceData object is able to store a set of properties into that database, which can be then queried using a user-built report.

The Cisco Collaboration Server only records Collaboration Server Session Platform SDK-based events for an application that is part of a Collaboration Server session. In stand-alone development mode, events are not recorded.

Configuration

Now that the features supported by Collaboration Server Session Platform SDK in the design of a collaborative application have been described, it is important to discuss how the Cisco Collaboration Server implements some of these features. Although a collaborative application is programmed entirely within the context of the application, there is a component on the server that must be associated with it to establish the limits and parameters of the application with regard to creation, performance, security, and feature availability.

The Cisco Collaboration Server presents a CSSP SDK configuration tool for establishing different collaborative application types and specifying their capabilities. Each type of collaborative application is associated with a specific Resource Type on the server. These Resource Types are set up and controlled by the Collaboration Server's administrator and the application's developer to describe everything from which object model the application uses to what types of events should be recorded to the database. The following are the configurable values for each Resource Type.

Resource class: The object model to be maintained on the server.

Resource page: The WWW page containing the application to be displayed in a browser upon session creation.

Participant, Session options: Restrict specific information that participants can see about other participants or the current session.

History options

Enable history: Checkbox to have the server maintain a history for this resource type.

Maximum history size: Maximum number of events to archive for the history.

Database options

Record session creation/termination: Record when the session was created and when it was terminated in the Collaboration Server database.

Record participation events: Record when the participants joined and left the session in the Collaboration Server database.

Record application events: Allow application-specific events to be recorded into the Collaboration Server database.

Event options

Maximum buffer size per participant: Maximum number of events to queue up for a client before purging and returning an error.

Leader options

Match Collaboration Session leader: Checkbox to have leadership determined from an external Collaboration Session. Only used for integration with a full Collaboration session.

Events inapplicable to leadership: Allow specific events from a non-leader participant to affect distributed state, even though that participant does not have leadership privileges.

Integration of Collaboration Server Session Platform SDK Objects

A collaborative application does not actually have many integration points with the objects provided by Collaboration Server Session Platform SDK. Although there are several types of adapters, only one will be used, and typically it will only be referred to on initialization. Similarly, each application will use only one object model, or Resource Class. This section describes exactly when it is appropriate for a collaborative application to interact with the CSSP SDK objects.

Click to read the following sections:

Initialization

Shared application data

Method calling

Event handling

Error handling

Initialization

The first thing that a collaborative application must do when communicating with the server is identify itself as a participant in an existing session. The Cisco Collaboration Server provides a browser-based mechanism for creating a session, and these identification values must be passed to the application from the Collaboration Server. The easiest way to do this is using JHTML to create a dynamic page containing an applet that is the collaborative application.

To initialize the application's connection to the server, the client must create an Adapter and call its initialization function. As mentioned above, each type of Adapter implements a different network communication strategy. The application chooses the most appropriate strategy given its environment and its purpose, and passes in the following values to initialize itself:

AppContextId: The identification value of this participant.

ResourceId: The identification value of the session to connect to.

Event handler: The object to receive ResourceEvent notifications from the Cisco Collaboration Server.

Subscription Model (optional): A filter to prevent all but the desired events from being sent to this client.

Once the adapter initializes successfully, the client is returned the Resource Class object that implements the Resource Type's distributed object model.

Shared application data

If the Resource Type uses any object model besides the pure event-based object model, the application must define its shared application data, or its set of ResourceData objects. There may be only one of these for an application or there may be a many - it is up to you to decide how complex the application's state representation should be. The developer must implement the serialization functions on its ResouceData objects, as well as implement any other functions that are necessary, such as merge functions, load/save functions, or database functions.

Method calling

Throughout a collaborative application, sections exist where local state changes. This state needs to be relayed to the distributed state on the Collaboration Server. At these points, the collaborative application calls methods on the object model to trigger the modification of the distributed state by the server.

Event handling

Type topic text here. As described above, a collaborative application must implement the IEventHandler interface to handle event notification from the Collaboration Server that the distributed state has changed. When Collaboration Server Session Platform SDK receives new events from the server, it calls a method on this interface, passing in a vector of new events. The application's implementation of this function should iterate through the new events and modify the application's local state based on the information contained in each event.

Error handling

Collaboration Server Session Platform SDK defines a set of errors that may occur during the course of a collaborative session. These errors can be thrown by the

server or the client system. They may occur in the application's stack or internally to CSSP SDK's event retrieval system. The IEventHandler interface defines a second function that must be implemented by the application to deal with errors outside of the application's stack, but other exceptions may be thrown by almost any CSSP SDK function call.

The following three general types of errors exist in the CSSP SDK architecture. Each type has specific codes that describe the actual error in more detail.

Communications Exceptions: These exceptions occur when the CSSP SDK client experiences an error communicating with the Collaboration Server, either due to a network error, corrupted data, or other similar problems.

Terminal Exceptions: These exceptions signify that something terminal has happened and the session may no longer continue. Examples include invalid identification, an already destroyed session, or an incorrect version of CSSP SDK.

Session Exceptions: These are errors that may arise in the normal course of a collaborative session. Examples include making a modification when the application does not have leadership privileges or trying to save a session's data when the Resource Type does not support saving.

Implementing a successful collaborative application

Creating the Application

This section of the document describes the steps that should be followed when writing a collaborative application. Of course, this is provided as an aid in designing and building an application using Collaboration Server Session Platform SDK, to the developer who is seeking to become more familiar with how to best use the CSSP SDK. Click to view the following sections:

Design collaborative usage of existing application

Determine format of shared data

Add a resource type and configure it

Implement shared data objects

Integrate application with CSSP SDK

Build and Test

Design collaborative usage of existing application

Before beginning any actual implementation work, some basic questions about the application's collaborative behavior must be answered. A collaborative application interacts differently with the user than traditional single-user applications. Bringing these questions to the forefront will help the developer make important decisions early on in the development cycle.

What collaborative behavior should exist?

This is the most obvious question. What information is shared between the remote instances of the application? What events from one instance should trigger activity on the others? It is a good idea to list exactly what portions of the application's functionality should be collaborative - that is, a complete description of how one instance should behave when another instance is modified.

How responsive should the application be?

Since your application will be operating over a network, it is important to keep in mind the frequency of distributed operations, and the latency associated with them. Remember that there may be several seconds of delay between when one instance sends an event and when another receives it - build this into your application. Different combinations of command batching, event filtering, leadership, and data capture can result in different run-time performances of the collaborative system. Decide what level of interactivity is needed between participants and design your integration appropriately.

What roles exist?

One of the key advantages to using Collaboration Server Session Platform SDK is the ability to attach different roles to different participants in the session. These roles may be as subtly different as enabling/disabling a feature or two, or as disjoint as two completely separate applications. Decide who will be using your application and what different roles to exist, if any. Depending on how different the roles are, you may choose to reuse the same application with different flags, or you may choose to actually implement two completely distinct applications that use the same shared data objects.

Determine format of shared data

Now that you know how you would like your collaborative application to behave, you can start designing the integration with Collaboration Server Session Platform SDK. Review the set of actions for each role of your application and how each will cause behavior in another instance. Take a look at how your application currently stores its information. Also look at what data is maintained throughout an instance of the application, and how often that data is modified. These will give you clues as to where to begin.

What is the object model?

As described earlier, CSSP SDK provides a set of object models to help store an application's distributed state. The distributed state of an application is maintained on the server mostly for synchronization purposes, but also for the benefit of participants entering midway into a session, and for reestablishing local state after encountering an error. When choosing the object model to use in your application, try to match it to the format that the local application uses to maintain state. Refer to the descriptions of the four object models above and choose the model that makes the most sense. Remember that the primary goal of the object model is an easy framework for packaging up and applying changes in local state. From this point forward, think of your application in terms of its distributed object model - the events it throws and the state, collection, or revision history it contains.

What are the shared data objects?

Let's drill one level deeper now. You've already decided on an object model that

stores some number of objects your application will be manipulating. The objects to be manipulated were most likely chosen by you because of the original design and organization of the application, or because they just made sense from a programming point of view. These objects that you've selected will conceptually become your shared data objects. The implementation details will be discussed later, but for now decide what attributes of these objects should be distributed. Not all the attributes will always need to be shared - some may be inferred from others, while still others may be maintained locally. For example, the position of a graphical element may be shared, since you want both application instances to render the object in the same relative location. Conversely, the font that it displays may be kept local, since you want the font to be chosen by the individual user and you can not guarantee that both instances will be able to render any specific font. Once you know what attributes on these objects need to be shared they are already mostly completed.

What events are needed?

Next let's solidify the events that your application will want to define and use. There are generally three types of events:

System-defined events (session termination, participants joining or leaving the session, changes in leadership)

User-defined events caused by changes to the object model, accompanied by the change (removing an object from a collection, adding a revision to a base object)

User-defined events that are accompanied by user-defined data (simple notification)

Review your application's desired collaborative behavior and associate each action with either a system-defined or a user-defined event type. User-defined events can be as general or as specific as you find appropriate. Remember that these events are indications that something has happened, so the easier it is for your application to decipher exactly what it needs to do to update its local state, the better.

What configurable features are needed?

Finally, decide if and how you want to use some of CSSP SDK's other features, such as database reporting, session history, session leadership, and participant or session information. You will use this information to configure your resource type on the server.

Add a resource type and configure it

It is now time to create a resource type on the Cisco Collaboration Server for your collaborative application. Your resource type will essentially instruct the server on exactly how to handle the resource. In configuring the resource type, the server will

form security rules for your application, both about who can manipulate the resource and how they manipulate it, and about the resource's limits to the server's computational resources such as disk space and memory usage.

Begin by opening a browser and going to the *<server name>/Cisco_CS/html/capi/capiconfig.jhtml* page. Here you will see a section labeled "Existing Resources" and a field to enter a new Resource Type. Type a name for your Resource and click the add button. You have just created your Resource Type, and now you may configure it.

Choose your object model from the list provided under the `Resource' sub-heading.

Enter the HTML page to push in the browser when someone joins your session under the `Resource' sub-heading. This page will most likely contain your collaborative applet.

If you want to use the session's history, check the `Enable History' box under the `History' sub-heading and put a value for the maximum history size in the area below it. The server will only maintain a history list of the size indicated in that box.

If you have memory limitations, are dealing with a large amount of data in each session, or expect a large number of simultaneous sessions on one server, you may want to decrease the maximum event buffer size under the `Events' sub-heading. If a participant's connection to the server is temporarily interrupted, the server will store that participant's events for a period of time until either the participant reconnects to the server or the server times that participant out. This value specifies a maximum number of events to maintain per participant, and if this value is exceeded, a reconnecting caller will receive an error and must handle the reconnect differently.

Decide what sets of events you would like recorded to the database, and check the appropriate check boxes under the `Database' sub-heading.

Under the `Load/Save' sub-heading, decide if you would like the server to support load and save operations. The session's history may also be loaded and saved. Check the appropriate checkboxes.

Under the `Leadership' sub-heading, decide if you would like leadership to be dependent on the Cisco Collaboration session. If this is a stand-alone session, this value is ignored.In some circumstances, you may want a participant's events to affect the object model even if that participant is not the leader. Normally, no participant but a leader if it exists may modify the objects in the object model. If this is the case, enter those event types' values in the list under the `Leadership' sub-heading.

Finally, choose what information about the session and each participant your application will have access to. If, for example, you uncheck the `Phone Number' box, then an application will not be allowed to get the phone number for a

participant in session. These options can be found under the `Session Information' and `Participant Information' sub-headings.

Once you have decided on all the relevant options for your resource type, push the `Save' button at the bottom of the page.

Implement shared data objects

At this point you are ready to begin some implementation of your collaborative application. Let's start with your ResourceData objects - the basis for your shared state. To use Collaboration Server Session Platform SDK objects in your application, you should import the Com.WebLine.Capi package into your code.

Derive from ResourceData.

First you must derive an object from CSSP SDK's ResourceData class. You have several choices here:

You may choose to use these objects directly in your application.

You may choose to have your application's object's maintain an instance of the derived ResourceData class internally to the object.

You may choose to implement mapping functions that, given your application object, can produce an appropriate ResourceData object, and vice versa.Whichever scheme you implement, your ResourceData objects should store the shared information that will be transferred between participants.

Provide a default constructor.

This is needed internally by CSSP SDK.

Implement data accessors for your application's use.

These are application-defined functions used for accessing the actual data packaged in your ResourceData object. Implement them as you see fit.

Implement read and write functions.

These functions are derived from the ResourceData base class and are used to serialize the object back and forth to the server. The write function should write the distributed state of your object to a given stream, and the read function should read it back in, in the same format. That is, if you called write on the object and gave it a ByteArrayOutputStream to write it's state to, you should be able to convert that byte

array to a ByteArrayInputStream and read the state in from the bytes that were written previously. You may find it convenient to use the read and write functions on the CAPIUtils class to help with your serialization.Implement merge, recordToDb, load, save if desired. If your object will be using the merge or database features of CSSP SDK, you must override the merge and recordToDb functions with merging and database recording functionality implemented, respectively. If you are using the load and save features of CSSP SDK, you may choose to override the load and save methods to provide specialized loading and saving functionality. Otherwise, load and save just use read and write, respectively.

Integrate application with Collaboration Server Session Platform SDK

Finally, let's integrate the object model with your application. The goal here is primarily to create or modify ResourceData objects based on actions in your application and send them to your distributed object model. Additionally, you must respond to events from the server.

Manipulate the object model.

Locate the points in your application where actions will trigger changes in the object model, and insert the appropriate object model functions there. When these functions are called, CSSP SDK will update the distributed state on the Collaboration Server and distribute events to all participants in the session.

Handle events.

To respond to changes in the distributed state, implement the handleEvents method of IEventHandler. This method will be called by the CSSP SDK system whenever the server notifies your client of changes in the distributed state. The events will come with a type indicating what happened, a sender indicating who initiated the change, and some ResourceData object for determining what actually changed. For example, if the change is a revision made to a base object on the server, the accompanying data is the revision itself. Your application should use this accompanying ResourceData object to update any local state either by replacing the counterpart, locally stored ResourceData object, or by extracting information out of the ResourceData object and applying to application objects. When a ResourceData object comes from the server, it contains a unique identification value that can be used to identify the object.

Handle Errors.

Finally, make sure you handle all appropriate errors with your application. The three types of errors - communication, terminal, and session - may usually be handled in groups. Communication errors may spawn some notification to the user and perhaps a connection retry after a short period of time, terminal errors may just indicate that the collaborative session is over, and session errors may just present a dialog box to

the user to be dismissed. Either way, errors provide an important way to give feedback to the user, which is a key element of writing a collaborative application.

Build and Test

At last your application is complete. When you compile your application, add the Com.WebLine.CAPI package to your classpath. Afterwards, place your application classes in a place that is available to both the server and a browser, and you can test your collaborative application.

Open the <server name>/Cisco_CS/html/capi/capisession.jhtml page.

Enter user information if desired.

Select your resource type.

Enter a keyword.

On second browser, select join session.

Enter same keyword

See how it works

Cisco Collaboration Server

HTTP, Socket, Embedded, Stateless adapter configuration

The Cisco Collaboration Server currently supports Collaboration Server Session Platform SDK clients that communicate over HTTP, sockets, or through stateless adapters. Through the CSSP SDK configuration tool, you can choose whether or not to allow HTTP or socket connections, and you can also choose a socket port to allow communication over.

Timeouts and timeout configuration

Currently the Cisco Collaboration Server has very short timeouts for participants in a session. If an application does not communicate with the server for more than one minute, the server will clean up that participant. Note that even if a user is not interacting with the application, the application is still communicating with the server. Communication stops when the applet is closed (for example the browser is shut down or the user visits another page), or a network error occurs. To re-attach to the session, the participant must explicitly join the session again. Once all the participants leave a session, the session will be cleaned up, and all its state will be lost.

Stand-alone Mode: Creating and joining a session

Although the most powerful use of Cisco Collaboration Server Session Platform SDK comes when it is integrated directly into a Cisco Collaboration Session, the Collaboration Server also supports a stand-alone mode for developing individual collaborative application sessions. The Collaboration Server provides a web page for creating and joining these sessions, which are catalogued by keyword. A session can be created by anyone simply by choosing a Resource Type and specifying a keyword to associate with it. Anyone can join that session simply by entering the keyword.

You can find the entry page for stand-alone CSSP SDK sessions at `<server name>/Cisco_CS/html/capi/capisession.jhtml`.

Examples

Examples

The following examples can be found in the Com.WebLine.CAPI.Sample package. Documented source files are included.

HelloWorld1: Getting Started, Sending Events

This example shows simply how to use the Event-based object model (IEventResource) to send events between participants. When a button is clicked by any participant, a "Hello World" message is displayed to all the participants in the session. A clear button will clear all the participants' displays.

See:

Com\WebLine\CAPI\Sample\HelloWorld1.java

html\capi\sample\HelloWorld1.jhtml

HelloWorld2: Using Subscription Filters, Getting Participant Information

This example is slightly more complicated that the previous example. Not only does it display the "Hello World" message, but also it indicates who sent the message (using getParticipantInfo), and it will not display a message sent by itself (using SubscriptionFilter).

Note that CAPI has been updated for Collaboration Server 5.0. Subscription filters can now be modified dynamically, not just at the time of creation. A new method called **updateSubscriptionFilters(SubscriptionFilter filter)** is available in the following adapter files:

- Adapter.java
- IAdapter.java
- BaseAdapater.java
- server\CAPIJHTMLAdapter.java
- server\CAPILocalAdapter.java
- server\CAPIServletAdapter.java

See:

Com\WebLine\CAPI\Sample\HelloWorld2.java

html\capi\sample\HelloWorld2.jhtml

HelloWorld3: Using Custom Shared Data, Using History

This final HelloWorld example sends a "Hello" message to the other participants in the session in one of several different languages available from a drop-down list box in the user interface (using ResourceData). Additionally, it displays statistics showing how many times a message of each language was received.

See:

Com\WebLine\CAPI\Sample\HelloWorld3.java

Com\WebLine\CAPI\Sample\HelloWorld3Applet.java

html\capi\sample\HelloWorld3.jhtml

Magnetic Poetry

Leadership, Using Collections, Batching Commands, Loading and Saving

This example displays a collection of tiles, each with a word, and allows participants in the session to reposition the tiles. The collection is maintained on the server using the Collection-based object model (IVectorResource). Participants can add and remove tiles from the application, and load and save the complete set of words and its arrangement (using load and save). When a participant moves a tile, the entire path of the tile can be seen on each participant's local application (using command batching). Finally, one participant can take the control of the board and prevent the others from moving tiles (using leadership).

NOTE: This example is not available in the alpha release of Collaboration Server Session Platform SDK, Version 0.5.

TicTacToe: Stateless and Remote Connection Strategies

This example implements a tic-tac-toe game, letting two users play tic-tac-toe against each other, and other participants watch the game as it progresses. It demonstrates the Single-object based object model (IStateResource). The game can be played through an applet either over an HTTP connection (HttpAdapter) or a socket connection (SocketAdapter), or through an HTML interface (JHTMLAdapter), and the connection type can be switched back and forth.

See:

Com\WebLine\CAPI\Sample\TicTacToeApplet.java

Com\WebLine\CAPI\Sample\TicTacToeData.java

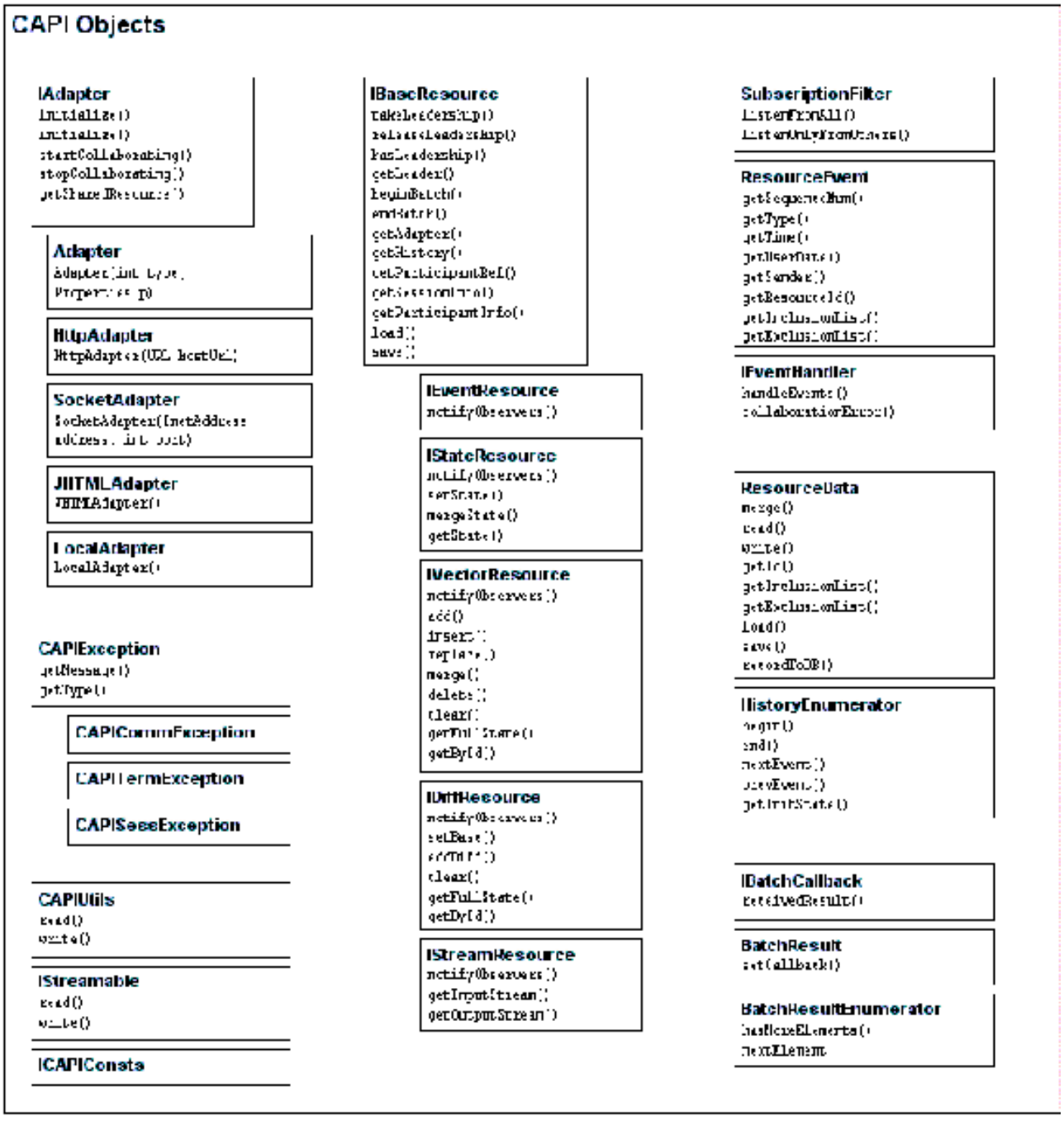
html\capi\sample\TicTacToeSel.jhtml

html\capi\sample\TicTacToe.jhtml

html\capi\sample\TicTacToeApplet.jhtml

The Cisco Collaboration Server Session Platform SDK

Below is a diagram of all the objects, and the names of their exposed methods, supported by CSSP SDK. For a complete description of all these objects, including method signatures, fields, and exceptions, please see the CSSP SDK Reference document included in this release.



Index

A

B

C

D

E

F

G

H

I

L

M

O

P

R

S

T

U