Avaya Conferencing Provider Interface (ACPI) User's Guide

Meeting Exchange 5.1 - 6.2

04-602742
5.1
November 2008
Issue 1

Notice

While reasonable efforts were made to ensure that the information in this document was complete and accurate at the time of printing, Avaya Inc. can assume no liability for any errors. Changes and corrections to the information in this document may be incorporated in future releases.

Documentation disclaimer

Avaya Inc. is not responsible for any modifications, additions, or deletions to the original published version of this Documentation unless such modifications, additions, or deletions were performed by Avaya.

Link disclaimer

Avaya Inc. is not responsible for the contents or reliability of any linked third party Web sites referenced elsewhere within this Documentation and Avaya does not necessarily endorse the products, services, or information described or offered within them. We cannot guarantee that these links will work all of the time and we have no control over the availability of the linked pages.

License

USE OR INSTALLATION OF THE PRODUCT INDICATES THE END USER'S ACCEPTANCE OF THE TERMS SET FORTH HEREIN AND THE GENERAL LICENSE TERMS AVAILABLE ON THE AVAYA WEBSITE AT http://support.avaya.com/LicenseInfo/ ("GENERAL LICENSE TERMS"). IF YOU DO NOT WISH TO BE BOUND BY THESE TERMS, YOU MUST RETURN THE PRODUCT(S) TO THE POINT OF PURCHASE WITHIN TEN (10) DAYS OF DELIVERY FOR A REFUND OR CREDIT.

Avaya grants End User a license within the scope of the license types described below. The applicable number of licenses and units of capacity for which the license is granted will be one (1), unless a different number of licenses or units of capacity is specified in the Documentation or other materials available to End User. "Designated Processor" means a single stand-alone computing device. "Server" means a Designated Processor that hosts a software application to be accessed by multiple users. "Software" means the computer programs in object code, originally licensed by Avaya and ultimately utilized by End User, whether as stand-alone Products or pre-installed on Hardware. "Hardware" means the standard hardware Products, originally sold by Avaya and ultimately utilized by End User.

License Type(s):

Concurrent User License (CU). End User may install and use the Software on multiple Designated Processors or one or more Servers, so long as only the licensed number of Units are accessing and using the Software at any given time. A "Unit" means the unit on which Avaya, at its sole discretion, bases the pricing of its licenses and can be, without limitation, an agent, port or user, an e-mail or voice mail account in the name of a person or corporate function (e.g., webmaster or helpdesk), or a directory entry in the administrative database utilized by the Product that permits one user to interface with the Software. Units may be linked to a specific, identified Server.

Database License (DL). Customer may install and use each copy of the Software on one Server or on multiple Servers provided that each of the Servers on which the Software is installed communicate with no more than a single instance of the same database.

Copyright

Except where expressly stated otherwise, the Product is protected by copyright and other laws respecting proprietary rights. Unauthorized reproduction, transfer, and or use can be a criminal, as well as a civil, offense under the applicable law.

Third-party Components

Certain software programs or portions thereof included in the Product may contain software distributed under third party agreements ("Third Party Components"), which may contain terms that expand or limit rights to use certain portions of the Product ("Third Party Terms"). Information identifying the copyright holders of the Third Party Components and the Third Party Terms that apply is available on Avaya's web site at: http://support.avaya.com/ThirdPartyLicense/

For full information, please see the complete document, Avaya Third Party Terms, Document number 04-601558. To locate this document on the website, simply go to http://www.avaya.com/support and search for the document number in the search box.

Warranty

Avaya Inc. provides a limited warranty on this product. Refer to your sales agreement to establish the terms of the limited warranty. In addition, Avaya's standard warranty language, as well as information regarding support for this product, while under warranty, is available through the following Web site:

http://www.avaya.com/support.

Avaya fraud intervention

If you suspect that you are being victimized by toll fraud and you need technical assistance or support, call Technical Service Center Toll Fraud Intervention Hotline at +1-800-643-2353 for the United States and Canada. Suspected security vulnerabilities with Avaya Products should be reported to Avaya by sending mail to: securityalerts@avaya.com.

For additional support telephone numbers, see the Avaya Web site: http://www.avaya.com/support

Trademarks

Avaya and the Avaya logo are registered trademarks of Avaya Inc. in the United States of America and other jurisdictions. Unless otherwise provided in this Documentation, marks identified by "®," "™" and "SM" are registered marks, trademarks and service marks, respectively, of Avaya Inc. All other trademarks are the property of their respective owners.

Document Information

For the most current versions of documentation, go to the Avaya support Web site: http://www.avaya.com/support

Contents

Chapter 1: Avaya Conferencing Provider Application Programming Interface

Introduction

This guide describes the Avaya Conferencing Provider Interface (ACPI), a Java programming interface.

Related documentation

See the following publication(s) for information referenced in this guide but not covered in detail:

- *Meeting Exchange® 5.1 Installing the S6200 and S6800 Audio Conferencing Servers*
- *Meeting Exchange® 5.1 Installing the S700/780 Audio Conferencing Servers*
- *Meeting Exchange™ 5.0 Bridge Control API (BCAPI) Guide*

Overview

The Avaya Conferencing Provider Interface (ACPI) is an object-oriented, programming interface, written in Java. The API can be used by any application that can make a Java call and supports Java version 5.0 or newer. ACPI consists of two components:

- The Application Programming Interface, also known as ACP API.
- Implementations of the API.

Currently only one implementation is available: ACP-MODAPI-IMPL which can be used to interface with:

- Meeting Exchange S700/780 Conferencing Servers
- S6200 and S6800 Media Servers

Important:
In this document, the term conferencing provider describes the conferencing or media server providing the conferencing capabilities.

Conferencing service providers or enterprises wanting to create applications which interface with the conferencing provider can use the ACPI. The ACPI provides an interface to develop applications that:

- provide moderators with real-time conference management capabilities for the automated and reservationless environment.
- provide participants with an alternative route for accessing the conference management capabilities that are exposed through the TUI.
- provide operators with the real-time conference management capabilities that are provided in the BridgeTalk application.

Through ACPI operations, developers can create applications that:

- create ad-hoc conferences.
- open a scheduled or demand conference using it's participant codes.
- manage conference recording and playback.
- manage conference security.
- manage Q&A and Polling.
- dial out participants.
- intercept and observer participants and conferences.
- manage sub-conferences.
- record and play recorded messages to participants.
- manage help and other queues.
- transfer conference participants.

The Avaya Conferencing Provider Interface provides the real time conference management capabilities previously available in BCAPI as well as exposing the additional real time conference management capabilities previously restricted to Operators using the BridgeTalk application.

Chapter 2: Avaya Conferencing Provider Application Programming Interface

Introduction

This chapter details the Avaya Conferencing Provider Application Programming Interface (ACP API) which allows for real-time conference control.

System requirements

ACP API has the following minimum system requirements:

- Java version 5.0 or newer
- Memory (TBD)
- Processor (TBD)

Required jar files

The following jar files are required on the class path for ACP API:

acp-api-*versionnumber*.jar

acpl-*versionumber*.jar

where *versionnumber* are the version numbers of each jar file.

Implementation packaging requirements

Each implementation of ACP will be accompanied with the following documentation:

- The URI syntax which the implementation supports.
- The feature support matrix which describes the ACP API features supported by the implementation.

- The verb availability and permissions matrix which describes which operations are supported on which objects and which authorizations are permitted to execute the operations.
- The list of required .jar files in order for the implementation to function.
- Any JVM specific notes.

Object model

Figure 1 shows the generic classes from which all the conferencing objects are derived. These classes can be summarized as follows:

Object	Description
ConferencingObject	This is the base class of all conferencing objects. It provides methods for discovering properties, permitted operations and execution of operations.
ConferencingObjectContainer	This is a conferencing object that contains other conferencing objects.
ConferencingObjectQueue	This is a conferencing object container that contains an ordered list of a specific class of conferencing objects.
Reference	This encapsulates a reference to a conferencing object that can be transferred between Java Virtual Machines and used to find the corresponding ConferencingObject instance (providing that the reference remains valid).
Operation	These are java beans that represent operations that can be executed on conferencing objects. They provide methods to allow for blocking until the operation has completed execution.
ConnectionSpecificResource	These are resources that do not change and cannot have operations performed against them, but are specific to the conferencing provider that is connected to. The possible values of connection specific resources can be obtained from the specific connection.

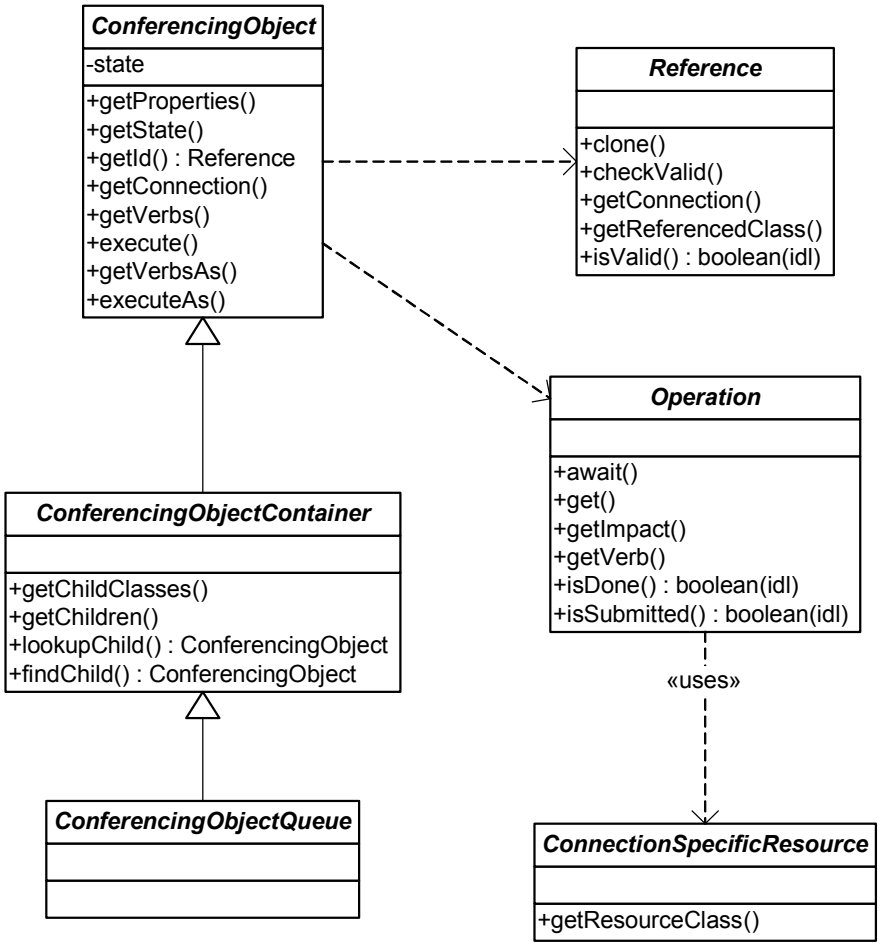

Figure 1: The generic base classes from which all the conferencing objects are derived

Figure 2 shows the main conferencing object classes. These classes can be summarized as follows:

Class	Descriptions
`Connection`	Represents the API connection to the conferencing provider.
`EndPoint`	Represents an EndPoint that has connected with the conferencing provider.
`Conference`	Represents a conference on the conferencing provider.
`EndPointQueue`	Represents a queue of EndPoints, for example EndPoints waiting for operator assistance, EndPoints waiting to ask a question in a question and answer session, etc.
`ConferenceQueue`	Represents a queue of conferences, for example conferences waiting for operator assistance.
`MediaSource`	A source of media that can be fed to EndPoints individually or to conferences, for example hold music.
`RecordPlaybackServer`	A device that can record and/or play back media.
`Operator`	Represents an operator. Operators can have an associated EndPoint which they can then use to interact with other EndPoints on the conferencing provider.
`Message`	Represents a media clips that are played to EndPoints as part of the call-flow that the EndPoint is in.
`CallFlowMatchingRule`	Represents a rule that assigns call flows to incoming connection requests from EndPoints.
`ParticipantCodeGroup`	Represents a set of participant codes within which each participant code uniquely identifies exactly one conference reservation at any one instant in time. A conferencing provider may support multiple participant code groups in which case there may be multiple conference reservations with the same participant code(s) on the conferencing provider. The participant code group can be used to differentiate these multiple conference reservations.
`OperationFactory`	Constructs operations which can subsequently be executed against conferencing objects.

Class	Descriptions
`MediaResourcePool`	Represents a resource pool that can be used for establishing media connections (e.g. EndPoints).
`MessageSet`	Represents a collection of Messages that will be used with a call flow.

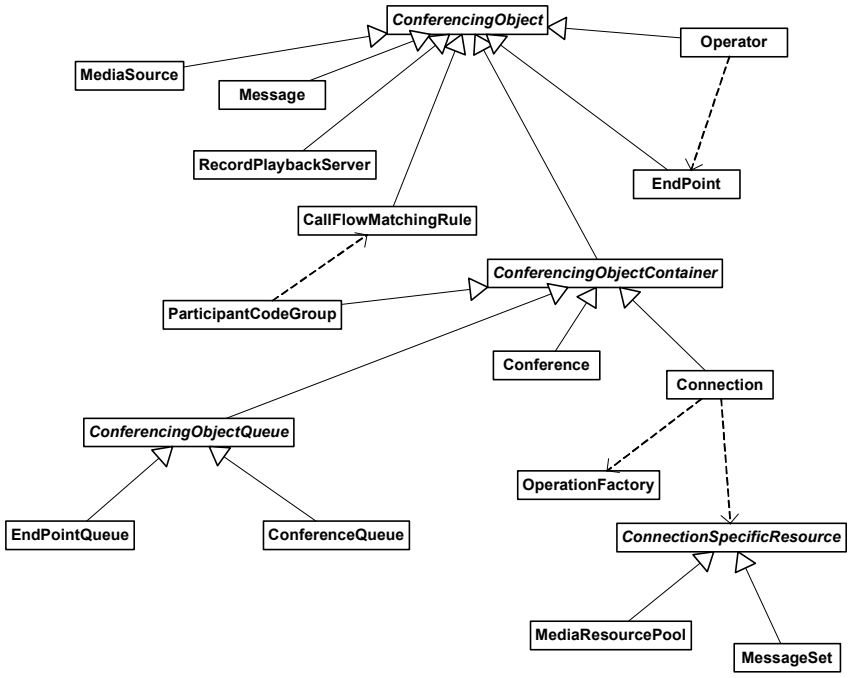

Figure 2: The main conferencing classes

Event model

Conference event handling is often complicated by the volume of events generated. A small conferencing provider system may not necessarily generate many events, however once a conferencing provider's capacity grows beyond approximately 4,000 simultaneous calls, peak event rates up to 1,000 events per second are not uncommon. The ideal event handlers should therefore be:

- Efficient
- Non-blocking
- Support parallel execution

The ACP API supports filtering of events through the use of specific event listeners, and registration of those listeners against specific conferencing objects. A listener will only ever receive events from the objects it has been registered with, thus while the conferencing provider may be generating multiple events, your application will only be notified of the events to which it has subscribed.

The ACP API assumes that all event listeners support parallel notification of events. For example, when an `EndPoint` enters a conference, a child added event will be notified from the conference. If the `EndPoint` disconnects as soon as it has entered the conference, a child removed event will then be notified. The ACP API will schedule the child added and child removed events in the order in which they occurred, however, the ACP API does not check to see if the child added event has finished processing before scheduling the child removed event. In fact, if the CPU resources are limited, the child added event may not have event started execution when the child removed event is scheduled. The Java Virtual Machine specification does not guarantee the order in which threads will be resumed, and as a result, it is entirely possible that the child removed event would be resumed prior to the child added event.

Note:
By default, the order of execution of event listeners is not guaranteed, but every event will be notified. An event listener should use event notifications to flag a dirty cache or the GUI displayed values as possibly being out of date and requiring updating.

If the order of events is critical, the ACP API can guarantee sequential event delivery for each source of events with a trade-off of agreeing a limit to the number of events pending notification. If an event listener must receive events in sequence, it should implement `SourceSequentialEventListener` in addition to any other event listener interfaces. Each instance of a `SourceSequentialEventListener` will be notified of all events for a specific source object using a single thread. Different source objects will be notified in parallel threads. If events are being notified faster than the notifications are being processed, the listener will receive warning events once the warning limit it agreed has been exceeded. If multiple warnings are ignored, the ACP API allows implementations to purge events to bring the backlog of events back under control. A `SourceSequentialEventListener` will be notified when events have been purged.

The following event listeners interfaces are supported by the ACP API:

Event listeners interfaces	Description
`AvailableVerbsChangeListener`	Which is notified of changes to the operations that are permitted on a source object.
`ChildListener`	A generic event listener which is notified when ConferencingObject instances are added to or removed from a ConferencingObjectContainer.
`ConferenceListener`	A specific ChildListener which is registered against a Connection and is notified when Conferences start (child added) and stop (child removed)
`ConferenceEndPointListener`	A specific ChildListener which is registered against a Conference and is notified when EndPoints join (child added) and leave (child removed) the conference.
`ConferencingObjectQueueListener`	Generic event listener which is notified when ConferencingObject instances are added to, reordered within, or removed from a ConferencingObjectQueue.
`EndPointQueueListener`	A specific ConferencingObjectQueueListener which is registered against an EndPointQueue and is notified when EndPoints join (child added), reorder within, and leave (child removed) the queue.
`DTMFListener`	Notified when DTMF events are received from EndPoints. If this is registered against an EndPoint it receives only those events generated by the EndPoint. If this is registered against a ConferencingObjectContainer, it will receive all DTMF events generated by the `EndPoint's` contained (directly or indirectly) within the ConferencingObjectContainer.
`StateChangeListener`	Notified when the state of a ConferencingObject changes.

Using complex conferencing features

This section details how to use some of the more complex conferencing features such as polling sessions, question and answer sessions, recording and playback of conferences, and accessing (also known as intercepting) conferences and `EndPoints`.

Polling sessions

Polling sessions allow participants to register votes using, for example, the DTMF keypad on a phone. A polling session is a variant on Lecture mode, all conferees are placed in the `OBSERVE_ONLY` state while moderators can contribute to the conference. This allows the moderators to, for example, inform all the conferees as to which DTMF keys correspond with the voting options, or inform the conferees how much longer the polling session will be active for.

Polling sessions are managed using one state:

State	Description
`POLLING_MODE`	Applies to a conference when it is in a Polling session.

And six operations:

Operation	Description
`PollingModeStart`	Starts a Polling session for a conference.
`PollingModeStop`	Stops a Polling session and returns the polling results as a map whose keys are the polling options which received votes and whose values are the number of votes.
`PollingQuery`	Retrieves either the current interim polling results, if in a polling session, or the polling results for the most recent polling session. Polling results are returned as a map whose keys are the polling options which received votes and whose values are the number of votes.
`PollingQueryDetail`	Retrieves either the current interim polling details, if in a polling session, or the polling details for the most recent polling session. Polling details are returned as a map whose keys are the **`EndPoints`** which voted and whose values are the votes.

Operation	Description
`PollingReset`	Clears the polling results, either for an individual `EndPoint` or for the entire conference.
`PollingAssist`	Puts an `EndPoint` into an access conference with the moderator or operator who executed the operation to determine what assistance is required.

Figure 3 illustrates a sequence diagram for a Polling session.

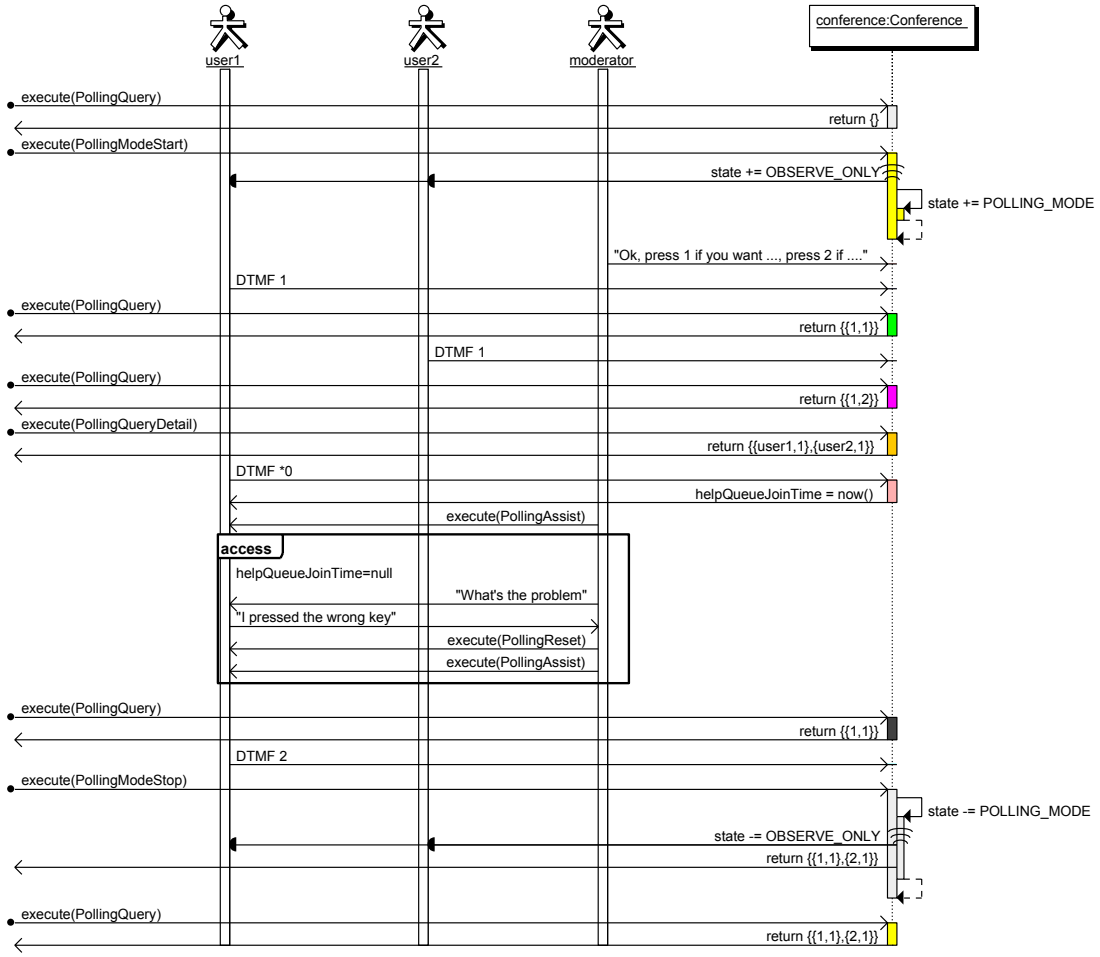

Figure 3: A high-level sequence diagram showing previous polling results being queried, a polling session being started, the moderator explaining the polling options, a user voting, the interim results queried, another user voting, the interim results and detail

being queried again, a user requesting assistance, the moderator providing assistance and resetting the user's polling choice, the interim results queried, the user voting again, the polling session being stopped, and finally the previous polling results being queried.

Question and Answer sessions

Question and Answer sessions are a variant on Lecture mode, i.e. all conferees are placed in the OBSERVE_ONLY state while moderators can contribute to the conference. In addition conferees can indicate that they have a question for the moderators (e.g. Meeting Exchange, by default, interprets the DTMF sequence *1 as indicating that the `EndPoint` has a question).

Conferees that have questions can be accessed, either in the order they registered their questions or in a random order, at which point they will have the OBSERVE_ONLY state removed while they ask their question. The conferee can then either have their question postponed, which puts them back into the queue of `EndPoints` with questions, or their question can be finished, which clears their QA_HAS_QUESTION state. In both cases, the OBSERVE_ONLY state is restored for the conferee. In between executing a QAFinishQuestion operation and executing a QANextQuestion operation, it may be possible to execute a QAPreviousQuestion operation which will retrieve the conferee against which the QAFinishQuestion operation was performed.

Q&A sessions are managed using two states:

State	Description
`QA_MODE`	Applies to a conference when it is in a Q&A session.
`QA_HAS_QUESTION`	Applies to an `EndPoint` when it has indicated that it has a question

A dedicated queue:

- Conference.getChildren(EndPointQueue.class, ConferencingObjectQueue.Purposes.QA_QUESTIONS.filter()) - note this queue is also returned by the QAModeStart operation

And six operations:

State	Description
QAModeStart	Which starts a Q&A session for a conference and returns the queue of questions.
QAModeStop	Which stops a Q&A session (can be executed against either the conference or the queue of questions).
QANextQuestion	Can be executed against either the queue of questions (for sequential questions in the order in which they were registered) or against EndPoints that have questions (for random access to questions).
QAPostponeQuestion	Can be executed against either the queue of questions or the EndPoint that is currently asking a question, in each case the EndPoint asking the question will be placed back into the queue.
QAFinishQuestion	Can be executed against either the queue of questions or the EndPoint that is currently asking a question, in each case the EndPoint asking the question will be removed from the queue of questions and their QA_HAS_QUESTION state will be cleared.
QAPreviousQuestion	Can be executed against the queue of questions while no conferee is asking a question and will retrieve the conferee against which a QAFinishQuestion operation was most recently performed. It is important to note that this operation inserts the conferee back into the queue of questions but does not unmute them.

Figure 4: A high-level sequence diagram showing a conference being put into Q&A mode, two participants indicating that they have questions via DTMF, the first question being retrieved from the queue, their question being postponed, then two questions being retrieved and completed followed by a recall of the last question before finally taking the conference out of Q&A mode.

Recording and playback of conferences

Conference recording and playback is controlled through three states:

State	Description
RECORDING_MODE	Indicates that the conference is being recorded.
PLAYBACK_MODE	Indicates that a recording is being played to the conference.
PAUSED	Indicates that the recording or the playback is temporarily paused.

Eight operations:

Operations	Description
RecordingModeStart	Starts recording.
RecordingModeStop	Stops recording
RecordingModePause	Temporarily pauses recording.
RecordingModeResume	Resumes a temporarily paused recording.
PlaybackModeStart	Starts playback of a recording.
PlaybackModeStop	Stops playback of a recording.
PlaybackModePause	Temporarily pauses playback of a recording.
PlaybackModeResume	Resumes playback of a recording that was temporarily paused.

And is restricted by the capabilities of the RecordPlaybackServer that is used. The allowed state transitions for the conference state are illustrated in Figure 5 below.

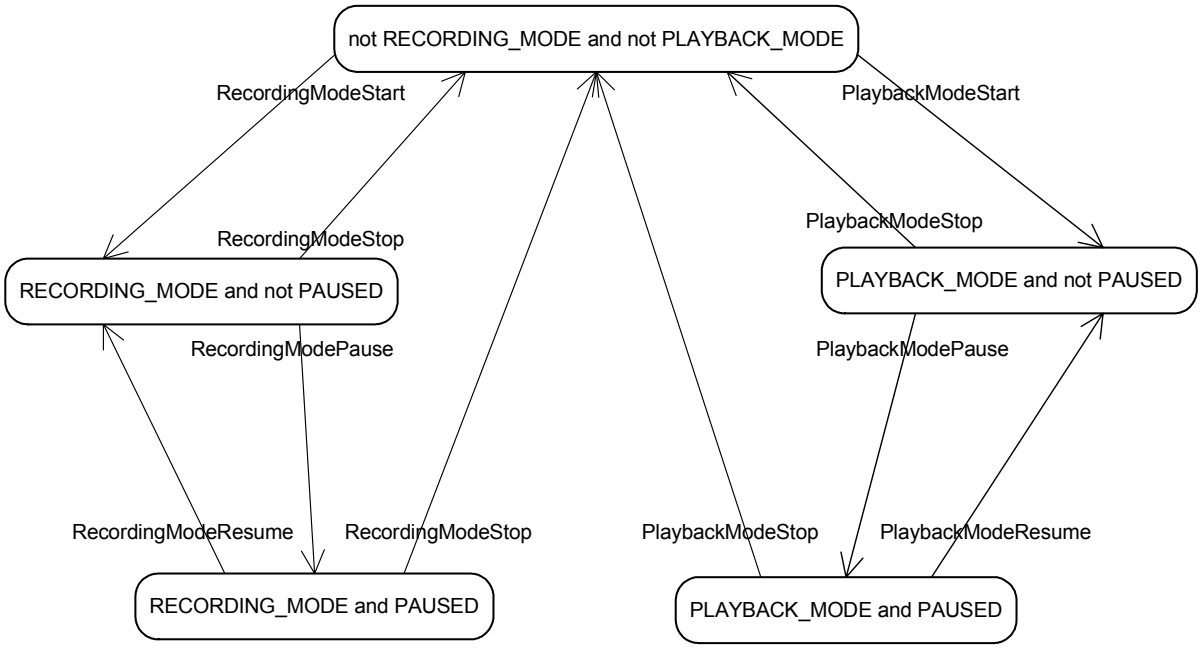

Figure 5: The Conference State transitions for recording and playback

The record-playback server used for a recording mode start operation must support recording (i.e. **`RecordPlaybackServer.isRecordingSupported() == true`**) in order to start recording, and similarly, playback must be supported by the record-playback server used for a playback mode start operation. The pause and resume operations are only available if the record-playback server supports pausing and resuming (i.e. RecordPlaybackServer.isPauseResumeSupported() == true). An application should query the conferencing provider to obtain a list of available record-playback servers and examine the supported features in order to select a record-playback server. If no recording-playback server is explicitly specified in the start operation, the conferencing provider is free to choose any of its available record-playback servers.

Accessing conferences and EndPoints

Operators often require interaction with conferences and conference participants. For example:

When a conference or a participant require assistance, the operator will need to talk to the conference or participant in order to determine the nature of the assistance required.

When a conference is running a question and answer session managed by an operator, the operator may want to talk to participants with questions in order to prioritize the next questions, or to get some details with which to introduce the questioner to the conference.

In order to interact with a conference or a participant, an operator must access that conference or participant first.

When an operator accesses a conference, the operator's **`EndPoint`** is placed into the conference in order to allow the operator to interact with the conference. Note that moderators do not need to access conferences as they (by definition) only have permissions for the conference that they are moderating.

When an operator (or a moderator) accesses a participant, a special temporary access conference is started and both **`EndPoints`** are moved into this temporary conference in order to allow them to interact without distracting the conference.

Accessing conferences and **`EndPoint`** is controlled through two states:

State	Description
`ACCESSED`	Indicates that the `EndPoint` or conference is currently being accessed.
`OBSERVED`	Indicates that the conference is currently being observed by at least one operator.

Note:
In some jurisdictions law enforcement agencies require the ability to observe telephony calls without the observed being able to detect the observation, therefore the OBSERVED state may not be reported for an `EndPoint`.

And two operations:

Operation	Description
`Access`	The authorization that is executing this operation will either: be placed into the conference that the operation is executed on; or will be placed into a temporary conference with the `EndPoint` that the operation is executed on.
`Observe`	The authorization that is executing this operation will have the media of the target instance directed to their `EndPoint`.

Figure 5 shows the state transitions for conferences and `EndPoints` when being accessed by an operator, while Figure Y shows the state transitions for a moderator accessing an `EndPoint`.

A special case of using an access conference is when a moderator wants to dial out a participant without distracting the main conference. This can be achieved with the following sequence of operations:

- Access the moderator's `EndPoint` as the moderator - this will move the moderator into a temporary conference.
- Make a call from the temporary conference.
- (Optionally) disconnects the call if the call is not required to join the conference.
- Access the moderator's `EndPoint` with as the moderator again to return the main conference.

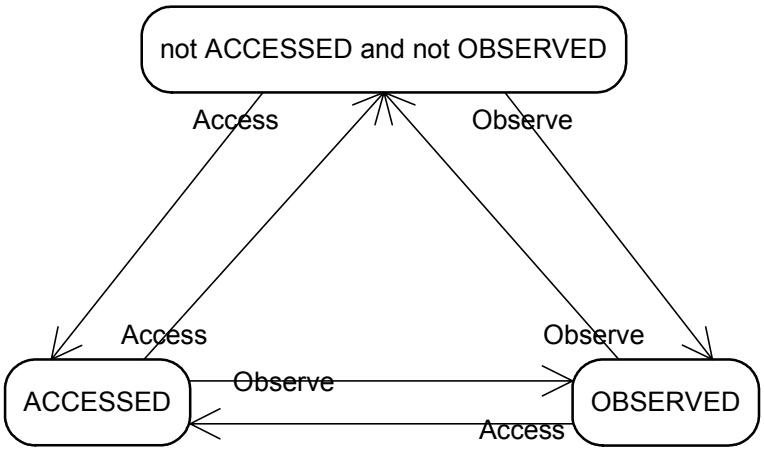

Figure 6: The Conference and EndPoint state transitions for access and observe by Operators.

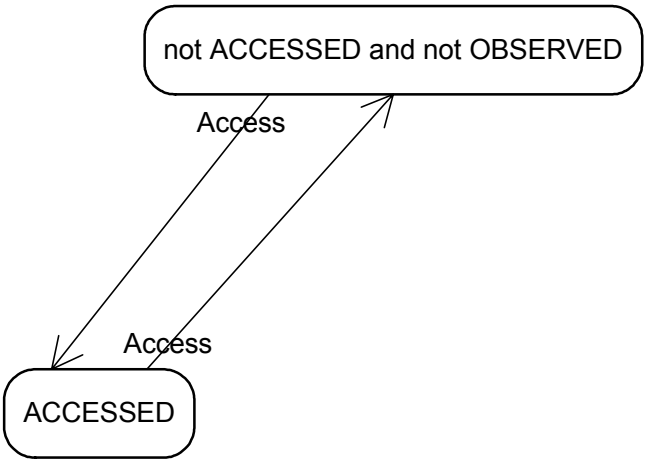

Figure 7: The EndPoint state transitions for access by Moderators.

Feature support matrix

The following table lists the features that are defined in the ACP API and which features must be supported by all ACPI implementations.

Feature	Must be supported
Classes	
EndPoint	Y
Conference	Y
EndPointQueue	N
ConferenceQueue	N
Operator	N
MediaSource	N
Message	N
MessageSet	N
Room	N
RecordPlaybackServer[1]	N
ParticipantCodeGroup	N
CallFlowMappingRule	N
Verbs	
MODIFY_DETAILS	Y
MAKE_CALL	Y
ACCESS	N
OBSERVE	N
DISCONNECT	Y
MODERATOR_DISCONNECT_MODE	N
AUTOMATIC_GAIN_CONTROL_MODE	N
SECURED_MODE	N

Feature	Must be supported
SECURITY_FEATURES	N
PROTECT_DETAILS	N
LECTURE_MODE	N
SILENCE	Y
MUTE	Y
HOLD	N
MODERATOR	N
OPEN	Y
CREATE	Y
REQUEST_HELP	N
CANCEL_HELP	N
ACCESS_NEXT_ENTRY	N
PLACE_IN_CONFERENCE	N
PLACE_IN_SUBCONFERENCE	N
PLACE_IN_MAIN_CONFERENCE	N
RECORDING_MODE_START[2]	N
RECORDING_MODE_PAUSE	N
RECORDING_MODE_RESUME	N
RECORDING_MODE_STOP	N
PLAYBACK_MODE_START[3]	N
PLAYBACK_MODE_PAUSE	N
PLAYBACK_MODE_RESUME	N
PLAYBACK_MODE_STOP	N
POLLING_MODE_START	N
POLLING_MODE_STOP	N
POLLING_RESET	N

Feature	Must be supported
POLLING_ASSIST	N
POLLING_QUERY	N
POLLING_QUERY_DETAIL	N
OPERATOR_LOGIN	N
OPERATOR_LOGOUT	N
PLAY_PARTICIPANT_NAME	N
PLAY_CONFERENCE_ROSTER	N
PLAY_MESSAGE	N
PLAY_AND_COLLECT	N
QA_MODE_START	N
QA_NEXT_QUESTION	N
QA_POSTPONE_QUESTION	N
QA_PREVIOUS_QUESTION	N
QA_FINISHED_QUESTION	N
QA_MODE_STOP	N
GET_CURRENT_TIME	Y
GET_RECORDING_URIS	N
DELETE_URI	N
MAKE_CALLS_FROM_LIST	N
CALL_LIST_GET	N
CALL_LIST_SET	N
CALL_LISTS_GET	N
MAKE_MEDIA_CONNECTION	N
FAULT	N
CONFERENCING_PROVIDER_LINK	N

1. Only those RecordPlaybackServers that have been configured on the bridge are available for use through the ACPI. If no analog recording lines or DRP has been set up on the bridge, recording/playback is not supported.

2. Only those RECORD_MODE classes that have been configured on the bridge are available for use through the ACPI. If no analog recording lines or DRP has been set up on the bridge, recording/playback is not supported.
3. Only those PLAYBACK_MODE classes that have been configured on the bridge are available for use through the ACPI. If no analog recording lines or DRP has been set up on the bridge, recording/playback is not supported.

Chapter 3: Avaya Conferencing Provider Meeting Exchange 5.1 Implementation

Introduction

The first available implementation of ACP API is ACP-MODAPI-IMPL. Modapi is a propriatary Avaya protocol used to control Meeting Exchange conferencing providers.

System requirements

ACP-MODAPI-IMPL has the following minimum system requirements:

- Java version 5.0 or newer
- Memory (TBD)
- Processor (TBD)

Supported conferencing providers

- Meeting Exchange 5.1 S700 / S780 Audio Conferencing Servers
- Meeting Exchange 5.1 S6200 / S6800 Audio Conferencing Servers

Required jar files

The following jar files are required on the class path for ACP API:

- acp-api-*versionnumber*.jar
- acp-modapi-impl-*versionnumber*.jar
- modapi-*versionnumber*.jar
- acpl-*versionnumber*.jar,

where *versionnumber* are the version numbers of each jar file.

URI syntax

The ACP-MODAPI-IMPL supports two URI schemes:

`modapi` for connection to S700/S780 Audio Conferencing Servers or for connection to S6200/S6800 Audio Conferencing Servers if the non-SSL version of the modapi protocol has been enabled.

`smodapi` for connection to S6200/S6800 Audio Conferencing Servers using the SSL encrypted version of the modapi protocol.

The syntax for the modapi protocol is:

`modapi://ip-or-name[:cmdport][;nat=on[;tcp=port;udp=port]]`

The syntax for the smodapi protocol is:

`smodapi://ip-or-name[:cmdport][;nat=on[;tcp=port;udp=port]]`

By default, the command port (**`cmdport`**) is 20002 for `modapi` and 20004 for `smodapi`. The default udp port for nat connections is 20008 while the tcp port is 20008 for modapi and 20010 for smodapi.

Some examples of valid URI's are:

`smodapi://mybridge.yourcompany.com`

`modapi://10.1.1.56;nat=on`

Feature support matrix

Feature	Implemented by ACP-MODAPI-IMPL
Classes	
`EndPoint`	✓
`Conference`	✓
`EndPointQueue`	✓
`ConferenceQueue`	✓
`Operator`	✓ [1]
`MediaSource`	✓
`Message`	✓

Feature	Implemented by ACP-MODAPI-IMPL
MessageSet	✓
Room	✓
RecordPlaybackServer	✓
ParticipantCodeGroup	✓
CallFlowMappingRule	✓
Verbs	
MODIFY_DETAILS	✓
MAKE_CALL	✓
ACCESS	✓
OBSERVE	✓
DISCONNECT	✓
MODERATOR_DISCONNECT_MODE	✓
AUTOMATIC_GAIN_CONTROL_MODE	✓
SECURED_MODE	✓
SECURITY_FEATURES	✓
PROTECT_DETAILS	✓
LECTURE_MODE	✓
SILENCE	✓
MUTE	✓[2]
HOLD	✓
MODERATOR	✓
OPEN	✓
CREATE	✓
REQUEST_HELP	✓
CANCEL_HELP	✓
ACCESS_NEXT_ENTRY	✓

Feature	Implemented by ACP-MODAPI-IMPL
PLACE_IN_CONFERENCE	✓
PLACE_IN_SUBCONFERENCE	✓[3]
PLACE_IN_MAIN_CONFERENCE	✓[4]
RECORDING_MODE_START	✓[5]
RECORDING_MODE_PAUSE	✓[6]
RECORDING_MODE_RESUME	✓[7]
RECORDING_MODE_STOP	✓[8]
PLAYBACK_MODE_START	✓
PLAYBACK_MODE_PAUSE	✓
PLAYBACK_MODE_RESUME	✓
PLAYBACK_MODE_STOP	✓
POLLING_MODE_START	✓[9]
POLLING_MODE_STOP	✓[10]
POLLING_RESET	✓[11]
POLLING_ASSIST	✓[12]
POLLING_QUERY	✓[13]
POLLING_QUERY_DETAIL	✓[14]
OPERATOR_LOGIN	✓
OPERATOR_LOGOUT	✓
PLAY_PARTICIPANT_NAME	✓
PLAY_CONFERENCE_ROSTER	✓
PLAY_MESSAGE	✓
PLAY_AND_COLLECT	✓
QA_MODE_START	✓[15]
QA_NEXT_QUESTION	✓[16]
QA_POSTPONE_QUESTION	✓ [17]

Feature	Implemented by ACP-MODAPI-IMPL
QA_PREVIOUS_QUESTION	✓[18]
QA_FINISHED_QUESTION	✓ [19]
QA_MODE_STOP	✓[20]
GET_CURRENT_TIME	✓
GET_RECORDING_URIS	✓
DELETE_URI	✓
MAKE_CALLS_FROM_LIST	✓
CALL_LIST_GET	✓
CALL_LIST_SET	✓
CALL_LISTS_GET	✓
MAKE_MEDIA_CONNECTION	✓
FAULT	✓[21]
CONFERENCING_PROVIDER_LINK	✓

1. However, operator instances are only available on the JVM where they were connected.
2. Available only if self mute is enabled in the system configuration.
3. Available only if sub conferencing mode is enabled in the system configuration.
4. Available only if sub conferencing mode is enabled in the system configuration.
5. Available only if recording has been enabled in the system configuration.
6. Available only if recording has been enabled in the system configuration.
7. Available only if recording has been enabled in the system configuration.
8. Available only if recording has been enabled in the system configuration.
9. Available only if polling feature is installed.
10. Available only if polling feature is installed.
11. Available only if polling feature is installed.
12. Available only if polling feature is installed.
13. Available only if polling feature is installed.
14. Available only if polling feature is installed.
15. Available only if Q&A feature is installed.
16. Available only if Q&A feature is installed.

17. Available only if Q&A feature is installed.
18. Available only if Q&A feature is installed.
19. Available only if Q&A feature is installed.
20. Available only if Q&A feature is installed.
21. available only if line faulting feature is installed

Verb availability and permissions matrix

The following table details the verb availability matrix.

	Conferencing object									
Operation	Connection	Conference	EndPoint	EndPointQueue	ConferenceQueue	MediaSource	RecordPlaybackServer	Message	Operator	ParticipantCodeGroup
Modify Details	✓		✓					✓		
Make Call		✓							✓	
Access		✓	✓							
Observe		✓	✓							
Disconnect		✓	✓			✓	✓ 1			
Moderator Disconnect Mode		✓								
Automatic Gain Control Mode		✓								
Secured Mode		✓								
Security Features		✓								
Protect Details		✓								
Lecture Mode		✓								
Silence			✓							
Mute		✓	✓							
Hold			✓							
Moderator			✓							

Open	✓									
Create	✓									
Request Help		✓	✓							
Cancel Help		✓	✓							
Access Next Entry				✓ 2	✓ 2					
Place In Conference			✓							
Place In Subconference			✓							
Place In Main Conference			✓							
Recording Mode Start		✓						✓		
Recording Mode Pause		✓								
Recording Mode Resume		✓								
Recording Mode Stop		✓						✓		
Playback Mode Start		✓						✓		
Playback Mode Pause		✓								
Playback Mode Resume		✓								
Playback Mode Stop		✓						✓		
Polling Mode Start		✓								
Polling Mode Stop		✓								
Polling Reset		✓	✓							
Polling Assist			✓							
Polling Query		✓								
Polling Query Detail		✓								
Operator Login	✓									

Operator Logout									✓	
Play Participant Name			✓							
Play Conference Roster			✓							
Play Message			✓							
Play And Collect			✓							
Qa Mode Start		✓								
Qa Next Question		✓	✓	✓ 2						
Qa Postpone Question			✓	✓ 2						
Qa Previous Question			✓	✓ 2						
Qa Finished Question			✓	✓ 2						
Qa Mode Stop		✓		✓ 2						
Get Current Time	✓									
Get Recording Uris	✓									
Delete Uri	✓									
Make Calls From List		✓								
Call List Get	✓									
Call List Set	✓									
Call Lists Get	✓									
Make Media Connection						✓	✓ 3			
Fault			✓			✓	✓ 3			
Conferencing Provider Link			✓							

1. Only available for analog, not digital.
2. Some, but not all queues support these operations.
3. Only available for analog, not digital.

Operation	Authorization	Connection	Anonymous Participant	Moderator same conference	Moderator other conference	Conferee same conference	Conferee other conference	Operator with EndPoint	Operator without EndPoint
Open	Connection	✓	✓		n/a		n/a	✓	✓
Create	Connection	✓			n/a		n/a	✓	✓
Operator Login	Connection	✓			n/a		n/a		
Get Current Time	Connection	✓	✓	✓	n/a	✓	n/a	✓	✓
Get Recording Uris	Connection	✓		✓	n/a		n/a	✓	✓
Delete Uri	Connection	✓			n/a		n/a	✓	✓
Call List Get	Connection	✓			n/a		n/a	✓	✓
Call List Set	Connection	✓			n/a		n/a	✓	✓
Call Lists Get	Connection	✓		✓	n/a		n/a	✓	✓
Modify Details	Conference	✓		✓		#3		#5	#5

Operation	Authorization	Connection	Anonymous Participant	Moderator same conference	Moderator other conference	Conferee same conference	Conferee other conference	Operator with EndPoint	Operator without EndPoint
Make Call	Conference	✓		✓				#6	#6
Access	Conference								#6
Observe	Conference								#6
Disconnect	Conference	✓		✓				#6	#6
Moderator Disconnect Mode	Conference	✓		✓					
Automatic Gain Control Mode	Conference	✓		✓				#6	#6
Secured Mode	Conference			✓					
Security Features	Conference	✓						#6	#6
Protect Details	Conference	✓						#6	#6
Lecture Mode	Conference	✓		✓				#6	#6
Mute	Conference			✓					

Operation	Authorization	Connection	Anonymous Participant	Moderator same conference	Moderator other conference	Conferee same conference	Conferee other conference	Operator with EndPoint	Operator without EndPoint
Request Help	Conference			✓		✓			
Cancel Help	Conference	✓		✓		✓		#6	#6
Recording Mode Start	Conference	✓		✓				#6	#6
Recording Mode Pause	Conference	✓		✓				#6	#6
Recording Mode Resume	Conference	✓		✓				#6	#6
Recording Mode Stop	Conference	✓		✓				#6	#6
Playback Mode Start	Conference	✓		✓				#6	#6
Playback Mode Pause	Conference	✓		✓				#6	#6

Operation	Authorization	Connection	Anonymous Participant	Moderator same conference	Moderator other conference	Conferee same conference	Conferee other conference	Operator with EndPoint	Operator without EndPoint
Playback Mode Resume	Conference	✓		✓				#6	#6
Playback Mode Stop	Conference	✓		✓				#6	#6
Polling Mode Start	Conference	✓		✓				#6	#6
Polling Mode Stop	Conference	✓		✓				#6	#6
Polling Reset	Conference	✓		✓				#6	#6
Polling Query	Conference	✓		✓				#6	#6
Polling Query Detail	Conference	✓		✓				#6	#6
Qa Mode Start	Conference	✓		✓				#6	#6
Qa Next Question	Conference	✓		✓				#6	#6

Operation	Authorization	Connection	Anonymous Participant	Moderator same conference	Moderator other conference	Conferee same conference	Conferee other conference	Operator with EndPoint	Operator without EndPoint
Qa Mode Stop	Conference	✓		✓				#6	#6
Make Calls From List	Conference	✓		✓				#6	#6
Modify Details	EndPoint	✓	#3	✓		#3		#7	#7
Access	EndPoint			✓					#7
Observe	EndPoint								#7
Disconnect	EndPoint	✓	#3	✓		#3		#7	#7
Silence	EndPoint	✓		✓				#7	#7
Mute	EndPoint	#1	#3	#1		#3		#7	#7
Hold	EndPoint	✓		✓				#7	#7
Moderator	EndPoint	#2		✓				#7	#7
Request Help	EndPoint		#3			#3			
Cancel Help	EndPoint	✓	#3	✓		#3		#7	#7

Operation	Authorization	Connection	Anonymous Participant	Moderator same conference	Moderator other conference	Conferee same conference	Conferee other conference	Operator with EndPoint	Operator without EndPoint
Place In Conference	EndPoint	✓						#7	#7
Place In Subconference	EndPoint	✓		✓		#4		#7	#7
Place In Main Conference	EndPoint	✓		✓		#4		#7	#7
Polling Reset	EndPoint	✓		✓				#7	#7
Polling Assist	EndPoint			✓					#7
Play Participant Name	EndPoint			✓		✓			#7
Play Conference Roster	EndPoint			✓		✓			#7
Play Message	EndPoint	✓		✓				#7	#7
Play And Collect	EndPoint	✓		✓				#7	#7

Operation	Authorization	Connection	Anonymous Participant	Moderator same conference	Moderator other conference	Conferee same conference	Conferee other conference	Operator with EndPoint	Operator without EndPoint
Qa Next Question	EndPoint	✓		✓				#7	#7
Qa Postpone Question	EndPoint	✓		✓				#7	#7
Qa Previous Question	EndPoint	✓		✓				#7	#7
Qa Finished Question	EndPoint	✓		✓				#7	#7
Fault	EndPoint	✓						#7	#7
Conferencing Provider Link	EndPoint	✓						#7	#7
Access Next Entry	EndPointQueue								✓

Operation	Authorization	Connection	Anonymous Participant	Moderator same conference	Moderator other conference	Conferee same conference	Conferee other conference	Operator with EndPoint	Operator without EndPoint
Qa Next Question	EndPointQueue	✓		✓				#8	#8
Qa Postpone Question	EndPointQueue	✓		✓				#8	#8
Qa Previous Question	EndPointQueue	✓		✓				#8	#8
Qa Finished Question	EndPointQueue	✓		✓				#8	#8
Qa Mode Stop	EndPointQueue	✓		✓				#8	#8
Access Next Entry	ConferenceQueue				n/a		n/a		✓
Disconnect	MediaSource	✓			n/a		n/a	✓	✓

Operation	Authorization	Connection	Anonymous Participant	Moderator same conference	Moderator other conference	Conferee same conference	Conferee other conference	Operator with EndPoint	Operator without EndPoint
Make Media Connection	MediaSource	✓			n/a		n/a	✓	✓
Fault	MediaSource	✓			n/a		n/a	✓	✓
Modify Details	Message	✓			n/a		n/a	✓	✓
Recording Mode Start	Message				n/a		n/a		✓
Recording Mode Stop	Message				n/a		n/a		✓
Playback Mode Start	Message				n/a		n/a		✓
Playback Mode Stop	Message				n/a		n/a		✓
Make Call	Operator	✓			n/a		n/a	✓	✓
Operator Logout	Operator	✓			n/a		n/a	✓	✓

#1 The Mute operation can be used to put an EndPoint into the MUTED state, but only an operator or the MUTED EndPoint's authorization is permitted to remove the MUTED state.

#2 The MODERATOR state can only be applied in conferences that can have moderators.

#3 These operations are only available for the EndPoint's authorization, i.e. an anonymous end point can disconnect itself but cannot disconnect any other EndPoints.

#4 These operations are only available for the EndPoint's authorization if subconferencing is enabled for participants on the conferencing provider, i.e. if Meeting Exchange is configured to allow anyone to move to a subconference via DTMF, then a conferee will be able to move their end point into a subconference.

#5 If the conference is neither secured nor has details protected.

#6 If the conference is not secured.

#7 If the conference that the EndPoint is in is not secured.

JVM specific notes

The system has been designed to work with any JSR-176 compatible Java Virtual Machine (JVM).

Note:
Avaya supports ACPI against the latest releases of versions 1.5.0 and 1.6.0 only of Sun's Java Virtual Machine.

The following container specific notes may be of use when using specific JVMs:

Sun Java Virtual Machines

By default, Sun's JVM has strong encryption enabled.

BEA JRocket Java Virtual Machines

By default, BEA's JRocket JVM has strong encryption enabled.

IBM Java Virtual Machines

In order to use the smodapi protocol, it is necessary to enable strong encryption in the IBM JVM.

1. Determine what Java version of IBM's JVM you are using, it must be at least Java 5.0 (JVM 1.5.0)
2. Go to the IBM Java Security information page (`http://www.ibm.com/developerworks/java/jdk/security/`)
3. Select the link for the JVM that you are using (either `http://www.ibm.com/developerworks/java/jdk/security/60/` or `http://www.ibm.com/developerworks/java/jdk/security/50/`)

4. Scroll down on the resulting page and click on the "IBM SDK Policy files" link.
5. This will take you to the "Unrestricted JCE Policy files for SDK 1.4" page. If you already have an IBM ID and password, click on the "Sign in" link. Otherwise, click on the "register now" link to create an ID.
6. On the Sign in page, supply your IBM ID and Password.
7. Select "Unrestricted JCE Policy files for SDK 1.4.2" and click on Continue. (at time of writing the 1.4.2 policy files are the policy files to use for Java 1.5 and 1.6).
8. Scroll down to the License portion of the resulting page and click on the View license link to see the licensing terms for the download.
9. If the licensing terms are acceptable, check "I agree" and click on the "I confirm" link. If the terms are not acceptable, you will not be able to enable strong encryption and should click "I cancel".
10. Click on the Download now link to download the unrestrict142.zip file.
11. Extract the local_policy.jar and US_export_policy.jar files from the unrestrict142.zip archive.
12. Place these two files in the <JAVA_HOME>/jre/lib/security directory, replacing the existing files with the same names.

Chapter 4: Avaya Conferencing Pluggable Logging

Introduction

ACPI is designed to be independent of the logging framework that is used. The Avaya Conferencing Pluggable Logging (ACPL) library is used to provide this independence. When ACPI is initially loaded, it looks for implementations of ACPL. If no alternative implementations are found, it will use `java.util.logging` as the logging framework. If an alternative is found, then the alternative will be used. If multiple alternatives are found, the logger used is undefined.

Avaya supplied implementations

In addition to the default implementation which directs all logging through java.util.logging, Avaya provides three alternative implementations of ACPL:

- acpl-log4j - which will direct all ACPI logging through Apache log4j (`http://logging.apache.org/log4j/`).
- acpl-simple-log - which will direct all ACPI logging through SimpleLog (`http://simple-log.dev.java.net/`).

Developing a custom ACPL implementation

If the logging library required is not in the above list, a custom implementation can be developed. An ACPL implementation is a jar file that contains:

- an implementation of the `com.avaya.conferencing.logging.LogManager` interface;
- an implementation of the `com.avaya.conferencing.logging.Logger` abstract class;
- an implementation of the `acpl-commons-logging`
- an implementation of the `acpl-slf4j`

- a /META-INF/services/com.avaya.conferencing.logging.LogManager file that contains the fully qualified name of the class that implements `com.avaya.conferencing.logging.LogManager`.

Chapter 5: Example applications

Introduction

The following sample applications are distributed with the ACPI:

- `acp-jsp-roster-display`
- `acp-jsp-operator-queue`
- `acp-jsf-roster`

Running the example applications

The example applications are written using version 2.5 of the Servlet Specification and require a compatible web container, for example:

- Glassfish version 2
- Jetty version 6
- Apache Tomcat version 6

Each sample application uses three context parameters to specify the connection details to the conferencing provider. The context parameters must be specified before deploying the sample applications. Context parameters are contained in the `/WEB-INF/web.xml` file in each sample application.

Building the example applications

The sample applications can be built using http://maven.apache.org. Prior to building the sample applications it will be necessary to either install the ACP jar files into your local repository, or deploy the ACP jar files into your corporate maven repository.

acp-jsp-roster-display

This is a web application using Java Server Pages as its presentation layer. It provides a simple roster view of a conference. It illustrates:

- how to safely store references to conferencing objects in the container's session such that if the web container is clustered, the references remain valid
- how to use a ContextListener to open a connection to the ConferencingProvider when the web application is deployed, close the connection when the web application is undeployed, and restore a connection if the connection has been lost

It consists of five JSP pages and a ContextListener. The ContextListener is responsible for opening and closing the connection. The connection is stored in the context, and an additional method is used to retrieve the connection from the context and check that the connection remains valid (re-establishing the connection if necessary).

The JSP pages consist of:

JSP Page	Function
`index.jsp`	Presents a form prompting for the conference details and submits the form to openRoster.jsp.
`openRoster.jsp`	Validates the conference details. If the details are correct, then the request is redirected to displayRoster.jsp, otherwise it is redirected to index.jsp.
`displayRoster.jsp`	Displays the conference roster. If the conference has not yet started, the request is redirected to pleaseWait.jsp. If the conference has finished, the request is redirected to rosterClosed.jsp. While the conference is active, the page is set to reload every three seconds.
`pleaseWait.jsp`	Reloads displayRoster.jsp after a three second delay.
`rosterClosed.jsp`	Redirects to index.jsp after a fifteen second delay.

acp-jsf-roster

This is a web application using Java Server Faces 1.2 and Facelets as its presentation layer. It provides a basic view of a conference with both conference and participant controls. It illustrates:

- how to serialize references to conferencing objects between the server and the client

- how to link a web session with a conferencing session by presenting a unique code on the web session that the conferencing user enters using DTMF on their phone
- how to execute operations against conferencing objects using the authorization of a specific conferencing user.

It consists of four Beans, a ContextListener (as used in the `acp-jsp-roster-display` example) and a helper class to convert References into a serialized form that can be embedded in an HTML page.

There are two ways to pass a reference to a web client in order to identify which reference an operation is to be performed against:

1. On the server side, generate a unique identifier for each conferencing object and pass the id to the client, using a map to decode the result.
2. Serialize and deserialize the reference.

The first technique encounters issues when running on a clustered web server, as each node in the cluster must generate the name unique identifier for each referenced object, however, it involves sending less data to the client.

The second technique, which is used in this example, has the advantage of ensuring that, irrespective of which node in a web server cluster, as long as a connection to the conferencing provider is available and the referenced object remains valid, the reference can be used to identify conferencing objects.

- The ReferenceHelper class serializes and deserializes a reference into a $Base_{16}$ encoded string. Real applications would most likely use a more efficient encoding and probably encrypt the serialized form with a cluster-wide salt to limit the ability and effectiveness of client-side attacks.
- The ConnectionManagerBean class is responsible for generating self-identification codes for users as well as exposing the Connection to the conferencing provider through JSP/JSF Expression Language.
- The LoginBean class is used by the login page.
- The UserBean class is used to associate a successful login with the web session.
- The RosterBean class is used to expose the conference roster through JSP/JSF Expression Language and exposes the conference operations as JSF actions.

acp-jsp-operator-queue

This is a web application using Java Server Pages as its presentation layer. It illustrates:

- how to safely store references to conferencing objects in the container's session such that if the web container is clustered, the references remain valid

- how to use a ContextListener to open a connection to the ConferencingProvider when the web application is deployed, close the connection when the web application is undeployed, and restore a connection if the connection has been lost.
- use a Javabean to hold form values
- How to dynamically determine what operations are supported on a conferencing object, and how to subsequently execute an unknown (at compile time) operation.
- It consists of twelve JSP pages and a ContextListener. The ContextListener is responsible for opening and closing the connection. The connection is stored in the context, and an additional method is used to retrieve the connection from the context and check that the connection remains valid (re-establishing the connection if necessary).

The JSP pages consist of:

JSP Page	Function
`index.jsp`	Presents a form prompting for the operator logon credentials and submits the form to connect-validate.jsp. If the current session was associated with a currently logged in operator, the operator is disconnected before displaying the page.
`connect-validate.jsp`	Populates a simple Java bean with the values from index.jsp checks if there is a connection to the conferencing provider and redirects to connect-login.jsp if there is, otherwise to no-connection.jsp. **Note:** The remaining jsp pages perform similar checks redirecting to different pages if the entry condition for the page is not satisfied.
`connect-login.jsp`	Attempts to authenticate the operator and disconnects any pre-connected EndPoint associated with the operator into the conferencing provider. If the operator is successfully connected then the page flow redirects to connect-call.jsp, otherwise an error is displayed and disconnect.jsp is called.
`connect-call.jsp`	Attempts to make a call to the the operator. When the operator's EndPoint is successfully connected then the page flow redirects to queue.jsp, otherwise the page reloads periodically.
`no-connection.jsp`	Reloads itself every 5 seconds while there is no connection available to the conferencing provider. Once a connection is available it redirects to index.jsp.

JSP Page	Function
`disconnect.jsp`	Logs the operator off the conferencing provider and disconnects the associated EndPoint.
`Queue.jsp`	Every 5 seconds this page redraws it self and lists all the participants who have requested an operator help request from their conference by pressing *0. The page displays a "Logout" button which directs to disconnect.jsp and a "Next help request" button when help requests are in the queue which directs to access-next.jsp.
`access-next.jsp`	Attempts to perform an AccessNextEntry operation on the help queue. If the operation was successfully then the accessed EndPoint is stored in the session and the page redirected to accessing.jsp, otherwise if all the help queues are empty, the page flow is redirected to queue.jsp in order to redisplay the queue. **Note:** The operation may fail as another operator may be accessing the same EndPoint at the same time.
`accessing.jsp`	Illustrates how to query the operations that can be performed on a conferencing object, and how to determine if an unknown (at the time of application development) operation has required parameters. If an operation is to be performed on the accessed EndPoint, the page flow directs to access-do.jsp, if the help request is satisfied, the page flow directs back to access-finisg.jsp, otherwise the page refreshes every 5 seconds.
`access-do.jsp`	Performs the operation passed as a query parameter and redirects back to accessing.jsp.
`access-finish.jsp`	Takes the accessed EndPoint out of the access conference and redirects to queue.jsp.

Chapter 6: Migration from BCAPI to ACPI

Introduction

This chapter covers migration from BCAPI, the previous Avaya API for real time conference control, to ACPI. In order to allow customers to migrate their Meeting Exchange 4.1 to 5.0 applications developed using BCAPI, the ACPI comes with a BCAPI legacy adapter that converts BCAPI's object, events and command model into ACPI's object, event and operation model. Due to limitations in the design of the BCAPI api, it is not possible to expose the new functionality provided in ACPI through BCAPI. Therefore customers wishing to access new APCI functionality must rewrite their conference control code to use ACPI.

Important:
BCAPI is a deprecated API.

Mixed environments

ACPI does not support connections to Meeting Exchange 4.1 or Meeting Exchange 5.0. During a rolling upgrade of Meeting Exchange 5.0 to Meeting Exchange 5.1, there are some API issues that must be addressed.

- The BCAPI implementation for Meeting Exchange 4.1 or Meeting Exchange 5.0 cannot talk to a Meeting Exchange 5.1 bridge using the SSL encrypted protocol.
- The BCAPI legacy adapter depends on ACPI which requires Meeting Exchange 5.1.

Therefore, while a mixture of Meeting Exchange 4.1, Meeting Exchange 5.0 and Meeting Exchange 5.1 bridges are being interfaced to, the BCAPI implementation for Meeting Exchange 5.0 must be used. This will require disabling the firewall rule on the Meeting Exchange 5.1 bridges that blocks the non-SSL version of the modapi protocol.

Once all the Meeting Exchange bridges have been upgraded to 5.1, replace the Meeting Exchange 5.0 BCAPI with the 5.1 BCAPI legacy adapter.

Once all the Meeting Exchange 5.0 BCAPI implementations have been replaced by the BCAPI legacy adapter, the firewall rule blocking the non-SSL version of the modapi protocol can be re-enabled.

BCAPI - ACPI terminology translation table

The following table lists the BCAPI terminology and the corresponding terminology used in ACPI:

BCAPI terminology	ACPI terminology
`BridgeFactory`	ConferencingProviders

`Bridge`	`ConferencingProvider`
	• representing the type of “bridge”
	`Connection`
	• representing the connection to the “bridge”

`Conference`	Conference
`Participant`	EndPoint **Note:** a future version of the ACPI may support association of multiple EndPoints with the participant that is using those EndPoints. If this is available there may be a Participant class that associates, e.g. the video EndPoint with the audio EndPoint.
`Reservation Group`	ParticipantCodeGroup
`CallBrandingTable & CallBrandingEntry`	CallFlowMappingRule
`Parameters`	Properties

BCAPI terminology	ACPI terminology

`BridgeListener`	ACPI uses a fine grained event model.
	`AvailableVerbsChangeListener`
	• for when the operations permitted change
	`PropertyChangeListener`
	• for when properties change.
	`StateChangeListener`
	• for when the state of an object changes.
	`ChildListener`
	• for when children are added/removed.
	`ConferenceListener`
	• for when conferences start and stop (or use ChildListener)
	`ConferencingObjectQueueListener`
	• for when a child moves in a queue of children.
	`ConferenceQueueListener`
	• for when conferences move in a queue of conferences (or use ConferencingObjectQueueListener).
	`EndPointQueueListener`
	• for when EndPoints move in a queue of EndPoints (or use ConferencingObjectQueueListener).
	`DTMFListener`
	• for when DTMF tones are detected.

`EnterQueue`	This is an EndPointQueue that has getPurposes().contains (Purposes.CONNECTIONS_ASSIST_REQUIRED)
`SelfMute`	Mute
`OperatorMute`	Silence
`Mute`	Observe_Only

BCAPI terminology	ACPI terminology
`requestOperator & requestConferenceWideOperator`	RequestHelp
`CancelOperatorRequest`	CancelHelp
`Participant.disconnect & Conference.closeConference`	Disconnect
`CurrentTimeMillis`	GetCurrentTime
`Hold`	Hold
`Lecture`	LectureMode
`Dial`	MakeCall
`Dial with intercept`	Access, then MakeCall
`interceptOff`	Access
`getLocalParticipantIds`	getEndPoints
`getConferenceIds`	getConferences
`makeModerator`	Moderator
`enableModeratorHangup`	ModeratorDisconnectMode
`setParameter & setParameters`	ModifyDetails
`getParameter`	getPropertyValue
`getParameters`	getPropertyValues
`setPromptSet`	ModifyDetails
`openConference`	Open
`Bridge.isConnected`	Connection.isOpen
`Conference.inService`	Conference.isActive
`Participant.isConnected`	EndPoint.isActive
`PlaceInConference & transferToConference`	PlaceInConference
`returnToConference`	PlaceInMainConference
`addToSubConf`	PlaceInSubconference

BCAPI terminology	ACPI terminology
PlayAndCollect	PlayAndCollect
pausePlayback	PlaybackModePause
continuePlayback	PlaybackModeResume
startPlayback	PlaybackModeStart
stopPlayback	PlaybackModeStop
playRoster	PlayConferenceRoster
playAnnunciator	PlayMessage
playParticipantInfo	PlayParticipantName
startPolling	PollingModeStart
stopPolling	PollingModeStop
viewPollingResults	PollingQuery
clearPollingResults	PollingReset
startQA	QAModeStart
stopQA	QAModeStop
askQuestionOnQAChan	QANextQuestion
resumeQAChan	QAPostponeQuestion
pauseRecording	RecordingModePause
continueRecording	RecordingModeResume
startRecording	RecordingModeStart
stopRecording	RecordingModeStop
Lock	SecuredMode
setSecurityAllowed	SecurityFeatures
subConference room	Room
PromptSet	MessageSet
Annunciator messages	Message
MusicSource	MediaSource

Chapter 7: Troubleshooting

MalformedURLException: Cannot find an implementation supporting the protocol xxx

This exception is thrown when the ACP API cannot find an implementation. There are two main causes for this exception:

1. A typographic error in the connection URL where the protocol has been specified incorrectly. The solution is to correct the connection URL.
2. A class-loading error preventing ACP API from finding any implementations.

   ACP-API uses the Java SPI mechanism (`http://java.sun.com/j2se/1.5.0/docs/guide/jar/jar.html#Service%20Provider`) to discover the implementations that are available.

   By default, ACP API will use the classloader `Thread.currentThread().getContextClassLoader()` to search for ACPI implementations. Alternatively, the class-loader can be specified by using the alternate ConferencingProviders.getConnection method which takes the class-loader to use as a parameter.

   Most JavaSE applications do not modify the `Thread.currentThread().getContextClassLoader()`, for this class of application, the solution to this problem is to ensure that all the required jar files for the ACPI implementation supporting the required protocol are on the class-path.

   JavaEE applications run within a JavaEE container. JavaEE containers are designed to host multiple applications at the same time and often control the class-loaders of applications in order to isolate the applications running inside the container. For this class of application, it may be necessary to consult the JavaEE container's documentation to determine the deployment descriptor(s) and/or container configuration settings that should be applied in order to ensure that all the required jar files for the ACPI implementation supporting the required protocol are on the class-path of the `Thread.currentThread().getContextClassLoader()`.

A connection is established however, no events are subsequently received. (modapi & smodapi protocols)

The modapi and smodapi protocols communicate over three network connections:

- A TCP connection from the client (from any unused port) to the conferencing provider (by default, to either port 20002 for modapi or 20004 for smodapi) which is used to send operations for execution to the conferencing provider.

- A TCP connection from the conferencing provider (from any unused port) to the client (by default, to the first unused port in the range 5020 - 5040, otherwise any unused port) which is used to receive events from the conferencing provider.
- A UDP connection from the conferencing provider (from 20008) to the client (by default, to the first unused port in the range 5020 - 5040, otherwise any unused port) which is used to receive active talker information from the conferencing provider.

Solutions:

1. Check the firewall settings on the client to ensure that the TCP & UDP connections from the conferencing provider are not blocked.
2. If the conferencing provider and the client are separated by a NAT router, enable NAT traversal mode, whereby all connections are established by the client. To enable NAT traversal mode append;nat=on to the connection URL.

Active talker information is not updated (modapi & smodapi protocols)

Active talker information is communicated over UDP (see previous problem). If the client and the conferencing provider are separated by a router that does not deliver UDP packets, then active talker information can never be received. Otherwise try the solutions for the previous problem.

Participants start talking, but there is a delay before receiving the active talker notification (modapi & smodapi protocols)

The software media server used in Meeting Exchange, by default, only updates the active talker information once every 2 seconds. If the default interval is not responsive enough, a configuration parameter (asnInterval in softMediaServer.cfg) file can be used to specify how often active talker notifications are updated (in milliseconds).