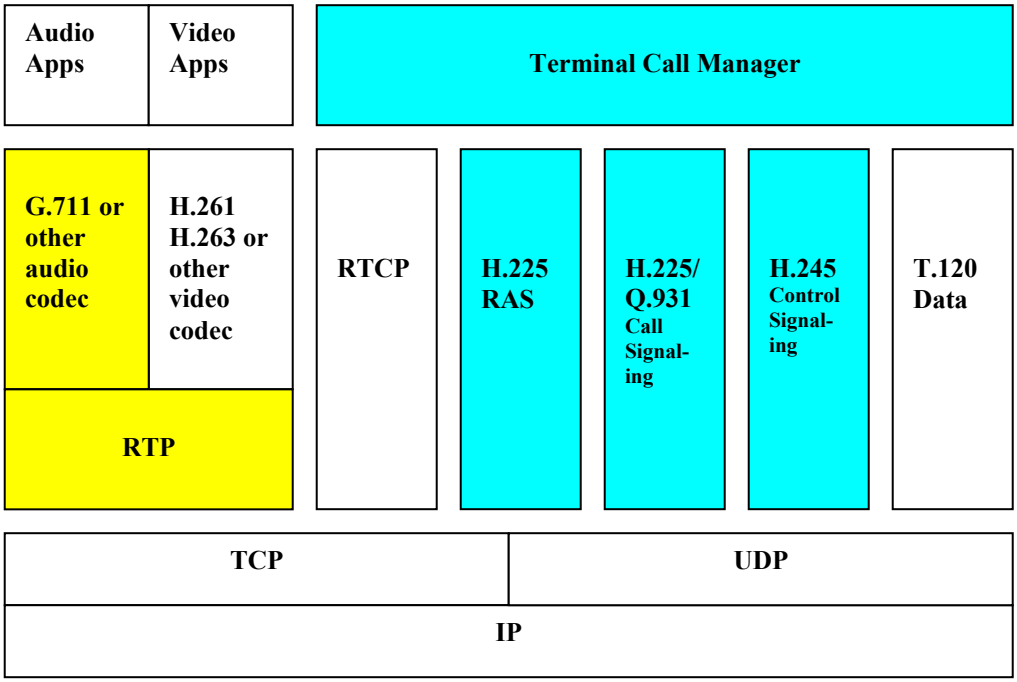
Objective Open H.323 for C User's Guide

Objective Systems, Inc. March 2005

Introduction

H.323 is an ITU-T Recommendation describing the protocols involved in establishing channels for multimedia communications over a packet-based network.

The Objective Open H.323 for C (ooH323c) protocol stack is an open source applications program interface (API) for building H.323 based applications. The stack implements Q.931/H.225 call signaling procedures, H.245 logical channel operations, and Registration, Admission, and Status (RAS) messaging for Gatekeeper communications. The supported parts of a standard H.323 stack are shown in the diagram below:



Key:

Supported by ooH323c

Supported by 3rd-party software

In this diagram, parts of the H.323 stack supported by ooH323c are shown in the light blue color. In addition, sample programs are included that contain third-party open-source media code to demonstrate a complete audio application that can play and receive a recorded audio file. The additional parts of the stack supported by this code are shown in yellow.

Architecture

The Objective Open H.323c for C stack is implemented as a simple, single-threaded application that allows a single H.323 endpoint to be set up that can create and teardown H.323 signaling and media channels. I/O multiplexing is done by means of a UNIX *select* or *poll* command to monitor all active I/O channels. TCP/IP and UDP communications are supported.

The main structure off which everything in the stack operates is the H.323 Endpoint – *OOH323EndPoint*. There is one global endpoint that is shared by all of the modules within the stack. This holds all of the standard configuration data (port numbers, terminal types, timeout values, etc). It also holds the active call list.

The active call list is a linked list of *OOH323CallData* records. Each time a call is initiated either from the stack or from a remote endpoint, a record is added to this list. This record is maintained throughout the life of the call. It contains details on negotiated capabilities and the different channels that have been set up to handle the call.

User application program interaction with the stack is accomplished by means of stack commands and callback functions.

Stack Commands

Stack command API functions are called to initiate a processing sequence within the stack. The typical initiation function that is called is *ooMakeCall* to initiate a call. Once this is done, callback functions are triggered as different H.323 messages are exchanged to set up the channels.

The following are the top-level stack commands:

Stack Command	Description
ooMakeCall	Initiate a call.
ooAnswerCall	Manually answer a call (this is only required when auto-answer is disabled).
ooHangCall	Terminate a call.
ooForwardCall	Forward call.

Callback Functions

The stack is event-driven and will react to I/O or timer events by calling registered callback functions. There are three types of callbacks, viz., H.323 callbacks (call level callbacks), H.225 callbacks (H.225 message level callbacks), and channel callbacks (callbacks to start/stop transmit/receive channels).

H.323 or Call Level Callbacks

Callback	Description
onNewCallCreated	This is triggered when a new call structure is created inside the stack for an incoming or outgoing call.

onIncomingCall	This is triggered when there is an incoming call. In case where a gatekeeper is being used, the new call must first be admitted by the gatekeeper.
onOutgoingCall	This callback is invoked after a Q.931 setup message is sent for an outgoing call.
onAlerting	This is triggered when a Q.931 alerting message is received for an outgoing call or when a Q.931 alerting message is sent for an incoming call.
onCallEstablished	This is triggered when a Q.931 connect message is sent in case of incoming call. In case of outgoing call, this is invoked when a Q.931 connect message is received. It is not invoked until after fast start and H.245 tunneling messages within the connect message are processed.
onCallForwarded	This is triggered when remote destination has forwarded our call to a new destination.
onCallCleared	This is triggered when a call is cleared.
openLogicalChannels	This is called when Master-Slave determination and Capabilities Negotiation procedures are successfully completed for a call.

H.225 or Message Level Callbacks

Callback	**Description**
onReceivedSetup	This is triggered when a H.225 SETUP message is received.
onReceivedConnect	This is triggered when a H.225 CONNECT message is received.
onBuiltSetup	This is invoked after a H.225 SETUP message is built and before it is sent out.
onBuiltConnect	This is invoked after H.225 CONNECT message is built and before it is sent out.

Channel Callbacks

These callbacks are used to start and stop transmit and receive media channels. User applications can register the same callbacks for various capabilities or can register different callbacks for different capabilities.

Callback	**Description**
startReceiveChannel	This is triggered either during a FastStart operation or in response to a received *OpenLogicalChannel* message from the remote endpoint.

startTransmitChannel	This is triggered either during a FastStart operation or in response to a received *OpenLogicalChannelAck* message from the remote endpoint.
stopReceiveChannel	This is triggered when a call is being cleared or in response to a *CloseLogicalChannel* request received from the remote endpoint.
stopTransmitChannel	This is triggered when a call is being cleared or in response to a *CloseLogicalChannel* request received from the remote endpoint.

Contents of the Package

The following diagram shows the directory tree structure that comprises the H.323 stack package:

```
ooh323c
 |
 +- doc
 |
 |
 +- src
 |  |
 |  +- h323
 |
 +- specs
 |
 +- tests
 |  |
 |  +- chansetup
 |  |
 |  +- simple
 |  |
 |  +- player
 |  |
 |  +- receiver
 |
 +
```

The purpose and contents of the various subdirectories are as follows:

- doc – this directory contains this document as well as the *H.323 Introduction*
- src – this directory contains all of the C source files
    - The top level src directory contains the C source files for the stack
    - The h323 subdirectory contains compiled ASN.1 code
- specs – this directory contains the H.323 ASN.1 specifications
- tests – this directory contains sample H.323 application programs.
    - The H.323 player application transmits audio data from a audio file (16 bit, 8000 samples/sec WAV file on windows, 16 bit raw data audio file on Linux)
    - The receiver application receives the RTP audio data stream and plays it on the speaker
    - The simple H.323 phone application demonstrates the ability to set up voice calls, negotiate capabilities and start voice channels.
    - The chansetup program contains basic skeleton code for setting up and tearing down an H.323 channel.

Installation and Build Procedures

The package is delivered as a .tar.gz or .zip archive file that may be unpacked into any root directory.

After unpacking the package, change directory to the package root directory.

Building the Package on Linux

Generate Makefiles for the package

./configure --prefix=<install-path>

The default <install-path> is /usr/local. This is where the final compiled oohh323c stack library is installed upon completion of the build and install procedure. Users must specify <install-path> to have the stack in their local user directory. For example, the following will cause the library to be installed in the user's ooh323c/lib subdirectory after 'make install' is executed:

./configure –prefix=$HOME/ooh323c

Build and Install the Package

To build the complete package including test applications:
./make

To build an optimized version (this is the default):
./make opt

To build the debug version:
./make debug

To install package,
./make install

This will install the libraries in the <install-path> specified while running 'configure' script.

Building the Package on Windows

The package includes Visual Studio 6 based workspace and project files.

1. Open the package root directory ooh323c-x.y, where x.y indicate the release number.

2. Open the ooh323c.dsw workspace, which includes all the projects within the package.

3. You can now do a batch build to build the complete package.

4. To build individual projects, dependencies are as follows:
   oostk.dsp – none
   oomedia.dsp – none
   h323ep.dsp - oostk.dsp, oomedia.dsp
   ooPlayer.dsp - oostk.dsp, oomedia.dsp
   ooReceiver.dsp - oostk.dsp, oomedia.dsp

   After a successful build the libraries will be installed in the ooh323c-x.y\lib\release and ooh323c-x.y\lib\debug directories.

Running the Sample Programs

Running the Programs on Linux

Running the Receiver and Player Applications (player and receiver)

Make sure that the path to the *liboomedia.so* shared object file is in your LD_LIBRARY_PATH path. This library is located in the installed *lib* subdirectory. If the library is in your local *ooh323c* directory tree, the following commands can be used to set the library path:

```
LD_LIBRARY_PATH=$HOME/ooh323c/lib
export LD_LIBRARY_PATH
```

note that in the command above, *ooh323c* may have additional version information appended.

Next, the 'receiver' application can be run as follows:

```
cd tests/receiver
./ooReceiver
```

Now, run the player application from a new console window as follows:

```
cd tests/player
./ooPlayer space.raw
```

The result should be the recorded sounds in the *space.raw* file being played on your computer's speakers.

Running the Simple Phone Application (simple)

Make sure all the path to the *liboomedia.so* shared object file is in LD_LIBRARY_PATH as specified in the previous section.

Next, change directory to the simple test directory:

```
cd tests/simple
```

To see the usage information including various options:
./simple OR ./simple --help

To make a call:

./simple [options] <remote>

where,
<remote> - is the dotted representation of the destinations IP address. In case of gatekeeper, aliases can also be used.

To receive a call:

./simple [options] --listen

You will find simple.log and media.log in the current directory.

Running the Programs on Windows

To run the sample programs on Windows, make sure that the media plug-in library *oomedia.dll* is in your PATH. The libraries are located in the ooh323c-x.y\lib\release and ooh323c-x.y\lib\debug directories.

First, run the receiver from the command prompt as follows (note: you must be in the package root directory):

```
cd tests\receiver\Release
ooReceiver.exe
```

A log file will be created in the current directory (ooReceiver.log). Also, a log file for the media plug-in will be created in the same directory (media.log).

Now run the player from the command prompt:

```
cd tests\player\Release
ooPlayer.exe states.WAV
```

A log file will be created in the current directory (ooReceiver.log). Also, a log file for the media plug-in will be created in the same directory (media.log).

To run the sample telephony endpoint application, again ensure that the media plug-in library (*oomedia.dll*) is in your PATH. The library is located in ooh323c-x.y\lib\release and ooh323c-x.y\lib\debug directories.

To run the telephony application:

```
cd tests\simple\Release
```

To see the usage information including various options:
simple OR simple --help

To make a call:

simple [options] <remote>

where,
<remote> - is the dotted representation of the destinations IP address. In case of gatekeeper, aliases can also be used.

To receive a call:

simple [options] --listen

You will find simple.log and media.log in the current directory.

Basic H.323 Application Design Pattern

Initializing the Endpoint

Before the ooh323c stack can do anything, the global endpoint structure must be initialized. This is accomplished by making the following function call:

```
ooH323EpInitialize ( args.. );
```

Arguments include information on the ID of the caller, type of call to be made (audio, video, or fax), and the name of a trace file where logging information should be written. For a complete argument list, please refer to the *ooH323c Run-time Reference Manual*.

Other properties can then be set through a series of "ooH323EpSet" calls. These set properties within the global endpoint object. For example:

```
ooH323EpSetAliasH323ID
```

can be used to set the H.323 ID within the alias address. See the *ooH323c Run-time Reference Manual* for the complete set of these functions.

Also, the use of "Registration, Admission, and Status" (RAS) gatekeeper client services should be initialized via a call to the following function:

```
ooGkClientInit (eGkMode, szGkAddr, iGkPort);
```

For simple point-to-point calls, the gatekeeper mode argument (*eGkMode*) should be set to *RasNoGatekeeper* and all other arguments may be set to zero. Other gatekeeper modes are *RasDiscoverGatekeeper* (discover a gatekeeper) or *RasUseSpecificGatekeeper* (use specific gatekeeper). The other arguments are used to set IP address and port numbers.

Add Capabilities

An H.323 application must specify to its peers what it is capable of doing. A capability negotiation will then take place within the H.245 message processing to arrive at a mutually agreed upon set of capabilities.

The application should add capabilities to the endpoint based on what it can support. For example, if an application can transmit and receive G711 ulaw encoded data, then it should add it to the endpoint as follows:

```
ooH323EpAddG711Capability (OO_G711ULAW64K, 30, 240, OORXANDTX,
                           &startReceiveChannel, &startTransmitChannel,
                           &stopReceiveChannel, &stopTransmitChannel);
```

The different types of capabilities are defined in *ooCapability.h*. These are constants that define known capability types. Users can extend this list if they plan to support additional capabilities not currently in this list.

Defining Callback Functions

Once the endpoint is initialized, the user should register callback functions they have defined. The Call Level callbacks can be registered as shown below:

```
ooH323EpSetH323Callbacks(h323Callbacks);
```

Here h323Callbacks is a structure of type OOH323CALLBACKS. All members of this structure should be initialized to either a valid callback value or NULL.

The H.225 message level callbacks can be registered as follows:

```
ooH323EpSetH225MsgCallbacks(h225Callbacks);
```

Here h225Callbacks is a structure of type OOH225MsgCallbacks. All members of this structure should be initialized by the user application either to a valid callback or to NULL.

The channel callbacks for starting/stopping receive/transmit media channels can be registered at the time of adding capabilities to the endpoint.

Create H.323 Listener

An H.323 listener is created to accept incoming connection requests. All that is required to start the listener is a call to the following function:

```
ooCreateH323Listener ();
```

This function takes no arguments; it just starts the listener service.

Initiating a Call

"Initiating a call" refers to the procedure to open channels for any type of media communications – not just audio. A call can be initiated to send video or data as well. The *ooMakeCall* function is used for this purpose. Its calling sequence is as follows:

```
ooMakeCall (dest, callToken, bufsiz, opts);
```

The following arguments are passed to this function:

dest – An identifier of the destination endpoint to be called. For example, an IP address and port.
callToken – A unique token identifier returned to identify the call
bufsiz – The size of the callToken buffer
opts – A reference to ooCalloptions structure which is defined in ootypes.h. This structure is used to override endpoint settings for a particular call. If non-NULL value is passed, the options specified in the structure will be applicable for the new call.

Closing a Call

Either side of an H.323 connection can terminate a call. The function used to do this is *ooHandCall*. Its calling sequence is as follows:

```
ooHangCall (callToken);
```

The following arguments are passed to this function:

callToken – The unique token identifier returned to identify the call. This was set in the call to ooMakeCall.

Shutting Down the Stack

A user can call the "ooStopMonitor" function from within a callback to shut the stack down. This will cause the "ooMonitorChannels" function to exit. Once this happens, the user should call to the "ooH323EpDestroy" function at the end of their program to do an necessary cleanup associated with the endpoint.