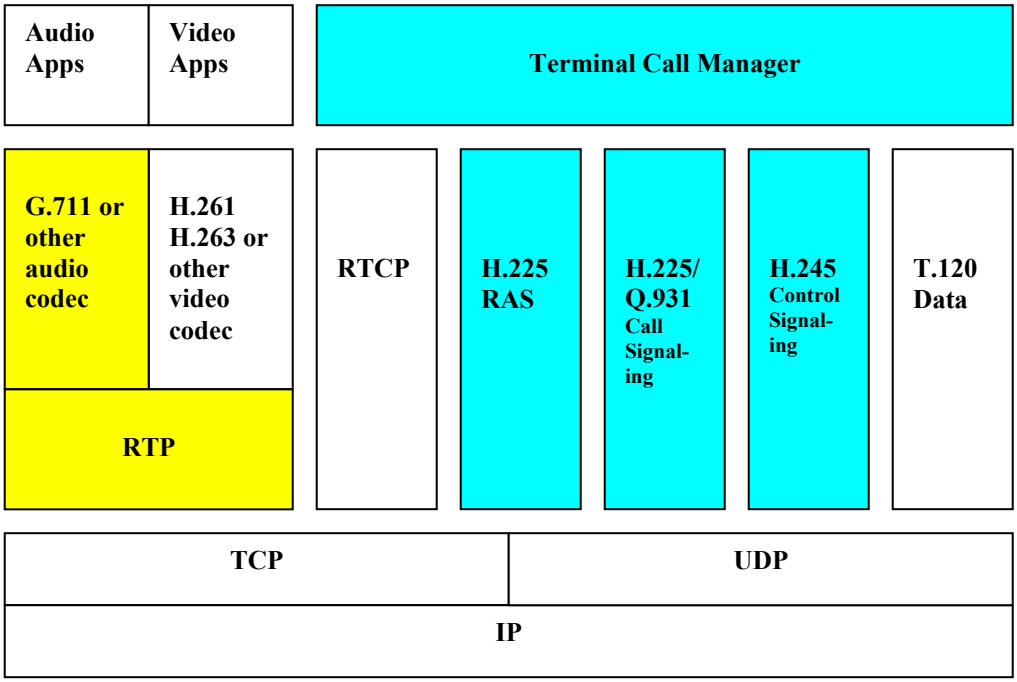
Objective Open H.323 for C User's Guide

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.

User application program interaction with the stack is accomplished by means of callback functions. The stack is event-driven and will react to I/O or timer events by calling registered callback functions. The following callbacks are defined for the various Q.931/H.225 message types:

Q.931/H.225 Message	**Callback**	**Description**
Setup	*onIncomingCall*	This is triggered when either a Q.931 setup message is received, or, if RAS is enabled, when the gatekeeper accepts the call setup request.
Setup	*onOutgoingCall*	This callback is invoked after a Q.931 setup message is sent.
Alert	*onAlerting*	This is triggered when a Q.931 alerting message is received.
Connect	*onCallEstablished*	This is triggered 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.
Facility	N/A	Handled internally. Tunneled H.245 messages or H.245 start requests are handled.
ReleaseComplete	*onCallCleared*	This is triggered when a Q.931 release complete message is received.
Information Notify Progress Status StatusEnquiry SetupAck	N/A	No action is currently taken on these message types.

The following callbacks are invoked within H.245 message processing code when logical channels are set up or torn down:

H.245 Message	**Callback**	**Description**
OpenLogical-Channel	*startReceiveChannel*	This is triggered after all H.245 capability exchange messages have been processed and a new channel is opened.
OpenLogical-ChannelAck	*startTransmitChannel*	This is triggered after a logical channel is successfully opened by the receiver and the acknowledge message is sent.

	stopReceiveChannel	This is triggered when a request to stop a logical channel is received. This can be caused by a number of different events including call termination by either endpoint or disconnect of the TCP/IP connection.
	stopTransmitChannel	This is triggered when a request to stop a logical channel is received. This can be caused by a number of different events including call termination by either endpoint or disconnect of the TCP/IP connection.

H.323 application programs use peer-to-peer communications for exchanging data. This means any program can act in a dual role as client or server. Further details on a basic design pattern are presented below in the Basic H.323 Application Design Pattern section.

Contents of the Package

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

```
ooh323c
 |
 +- doc
 |
 +- lib
 |
 +- 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*
- lib – this directory is created after 'make install' is executed. It contains the static and dynamic object libraries.
- 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
./simple
```

This provides a menu to make a call, accept incoming calls, or hang-up a call.

NOTE: For testing against ohphone, make sure -T -F -n options for ohphone are enabled. This ensures that ohphone does not use tunneling and faststart, which we do not support in this release.

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\h323ep\Release
```

To make a call:

```
simple.exe <dest-ip>
```

where <dest-ip> is the dotted representation of the destinations IP address.

To receive a call:

```
simple.exe
```

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

For testing against *ohphone*, make sure the -T -F -n options are enabled. This ensures that *ohphone* does not use tunneling and faststart. It is possible to use either of these items with ooH323c, however, no examples exist at this time showing their use.

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.

Capabilities are specified using a series of the following function call:

```
ooAddCapability (capability, type, callbacks);
```

The values for the *capability* argument 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.

The *type* argument specifies the capability type as defined in the choice tag constants in *MULTIMEDIA_SYSTEM_CONTROL.h* under the 'Capability' heading.

Callbacks are the various callback functions that should be invoked when a logical channel is opened or closed matching the given capability. The following callbacks may be specified here:

```
startReceiveChannel
startTransmitChannel
stopReceiveChannel
```

```
stopTransmitChannel
```

Defining Callback Functions

Once the endpoint is initialized, the user should register callback functions they have defined. Callbacks can be registered at both the H.245 level when logical channels are opened or closed (see the Add Capabilities section above) or at the Q.931/H.225 level when certain signaling messages are exchanged. The stack will invoke these functions when the events tied to the callbacks occur.

Q.931/H.225 callbacks are registered by calling the following function:

```
ooH323EpRegisterCallbacks
```

The user, at a minimum, will typically need to implement the capabilities callbacks to start and stop the media streams for their particular application. Starting and stopping RTP streams is currently not supported in the ooH323c stack, but third-party open source code is provided in some of the test programs that can be used for this purpose.

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);
```

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

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.