H.323 Stack/Framework
User's Guide

Objective Systems, Inc. October 2004

Introduction

H.323 is an ITU Recommendation describing the protocols involved in making a call with multimedia capabilities over a packet-based network without guaranteed QoS. H.450 is a series of supplementary service Recommendations, defining signaling and procedures used to provide telephony like services.

The H.323 framework stack is an open source stack for building H.323 based applications. This package consists of three main components as described below:

1. Framework of PER Encode/Decode API functions for encoding and decoding messages from the H.323 and H.450 ASN.1 specifications using the Packed Encoding Rules (PER) as defined in ITU standard X.691. The Objective Systems ASN1C compiler was used to generate the initial structures and encode/decode functions.
2. Stack built using the above framework. The stack currently has implementation for opening H.225 channel and H.245 channel. Though the stack currently supports handling of selective messages from H.2250 and H.245 family, it can be easily extended to add support for other messages using the underlying framework provided. In addition, stack supports plug-in architecture for media support, i.e., users can develop their own plug-in for RTP and audio handling.
3. Media plug-in, a sample plug-in library provided by Objective Systems, which supports audio and RTP data handling.

The figure below shows the layered architecture view of the various components of the stack

Audio Application
H.323 stack API + framework
Audio API
Socket Layer
Audio Plug-in

The final package is comprised of ANSI standard C code and/or binary libraries that can be ported to a wide-range of operating environments (including embedded).

Contents of the Package

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

```
ooh323c
 |
 +- doc
 |
 +- src
 |  |
 |  +- h323
 |  |          Framework component
 |  +- h450
 |
 +- specs
 |
 +- tests
 |  |
 |  +- simple
 |  |
 |  +- player
 |  |
 |  +- receiver
 |
 +- media(Plug-in library supporting RTP and audio functionality)
```

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 stack
    - The h323/h450 directories contain the framework component
- specs – this directory contains the H.323 ASN.1 specifications
- tests – this directory contains three sample H.323 applications.
    - 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 H.323 phone application demonstrates the ability to set up voice calls, negotiate capabilities and start voice channels.
- media – this directory contains Objective Systems audio and RTP plug-in library

Getting Started

The package is delivered as a .tar.gz or .zip archive file that may be unpacked into any root directory. After untaring the package, change directory to package root directory.

Building Package on Linux

Generate Makefiles for the package

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

The default <install-path> is /use/local. The only components installed are the ooh323c library, liboostk and the media plug-in library, liboomedia. (Both static and shared versions are installed)

Build and install Package

To build the complete package including test applications:

./make

To build optimized version:

./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.

Running Receiver and Player applications

Make sure that the installed libraries, liboostk.a and liboomedia.so are in your LD_LIBRARY_PATH path. First run the 'receiver' application as follows

cd tests/receiver
./ooReceiver

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

cd tests/player
./ooPlayer space.raw

Running phone application

Make sure all the libraries, i.e, liboostk.a, liboomedia.so are in your path.

cd tests/simple

To make a call:

./simple <dest-ip>

where dest-ip is dotted IP address of the remote H.323 endpoint.

To receive a call:
./simple

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.

Building 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
   simple.dsp - oostk.dsp, oomedia.dsp
   ooPlayer.dsp - oostk.dsp, oomedia.dsp
   ooReceiver.dsp - oostk.dsp, oomedia.dsp

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

6. To run examples, make sure that the media plug-in library oomedia.dll is in your PATH. The libraries is located in ooh323c-x.y\lib\release and ooh323c-x.y\lib\debug directories.

   Now run the receiver from command prompt:
   From package root directory ooh323c-x.y

   >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 plugin will be created in the same directory (media.log).

   Now run the player from command prompt:
   From package root directory ooh323c-x.y
   >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 plugin will be created in the same directory (media.log).

7. 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:
   From package root directory ooh323c-x.y
   >cd tests\simple\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 simple.log and media.log in the current directory.

8. 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.

H.323 stack component

H.323 stack component currently supports messages from H.2250 and H.245 necessary to setup a call and create audio channels. It leverages the H.323 Framework component to create a high level stack API. The main source files of this component and their functional description is given below:

File	Description
oo.c	Includes trace logic.
oochannels.c	Includes logic to create H.2250/H.245 channels and monitor them. This layer is built over socket layer. It also includes logic to read/write on channels and accept incoming connections.
ooh245.c	Includes logic to create and populate H.245 messages. This leverages the framework and provides a higher level stack API.
ooq931.c	Has logic to create and populate H.2250 message structures. Leverages the framework code and provides a higher level stack API.
ooSocket.c	Provides socket layer functionality
oosndrtp.c	Integrates sound and RTP plug-in component into the stack.
ootypes.h	Has definitions of common types. Defines ostkAppContext structure which holds all the important information for the current call.
ooCalls.c	Includes function to manage calls.
ooStackCmds.c	Includes function to Make Call/ Hang Call/ Close Stack
ooh323ep.c	Includes functions to create an endpoint. This includes setting configurations such as faststart, tunneling, coutry code, product id etc and adding capabilities to the endpoint.

Test Applications
The test applications are provided to demonstrate how to use the stack component. Here Player and Receiver applications are described.

Player

1. Creates an H.323 endpoint using *ooInitializeH323Ep*(). This function creates an endpoint inside stack which has important config information such as logfile name, terminal type, faststart and tunneling options, call type, product id, version, call-id to be generated for outgoing calls, caller name to be used for outgoing calls etc. This information is used by the stack to handle various H.225 and H.245 messages.
2. Next, it uses *ooH323EpRegisterCallbacks()* to register callbacks to be used for various events such as incoming-call, outgoing-call placed, call-established, call-cleared etc.
3. Next, it uses *ooAddAudioCapability()* to add audio capabilities to the endpoint.
4. Next, it loads the media plug-in using *ooLoadSndRTPPlugin()* function call.
5. At this point the application is ready to issue a MakeCall command to the stack. It uses the ooMakeCall() command and passes it the destination ip, destination port and buffer for callToken. On Return, the callToken buffer is populated with the unique identifier for the new call.
6. Now the application calls ooMonitorChannels(), to start the monitoring of the channels by the stack.

Receiver

Same as for *Player,* except *ooMakeCall()* is not called and instead *ooCreateH323Listener()* is called to listen for incoming calls.

H.323 Framework component

This component consists of source files under src/h323 and src/h450 directories. This component defines structures for representing all H.323 messages under H.323 protocol umbrella and logic to encode/decode them using PER encoding rules. Following section describes procedure to encode different types of H.323 messages using framework component.

Encoding a RAS Message

RAS (Registration, Admissions, and Status) messages are one class of H.323 messages. These messages are used in cases where H.323 endpoints within a H.323 zone are moderated by a H.323 Gatekeeper. RAS messages occur between endpoints and a gatekeeper to provide registration, admission, and status functions for endpoints within zones. The ASN.1 definition for a RAS message is as follows:

```
RasMessage ::= CHOICE
{
      gatekeeperRequest             GatekeeperRequest,
      gatekeeperConfirm             GatekeeperConfirm,
      gatekeeperReject              GatekeeperReject,
      registrationRequest           RegistrationRequest,
      registrationConfirm           RegistrationConfirm,
      registrationReject            RegistrationReject,
```

```
        unregistrationRequest           UnregistrationRequest,
        unregistrationConfirm           UnregistrationConfirm,
        unregistrationReject            UnregistrationReject,
        admissionRequest                AdmissionRequest,
        admissionConfirm                AdmissionConfirm,
        admissionReject                 AdmissionReject,
        bandwidthRequest                BandwidthRequest,
        bandwidthConfirm                BandwidthConfirm,
        bandwidthReject                 BandwidthReject,
        disengageRequest                DisengageRequest,
        disengageConfirm                DisengageConfirm,
        disengageReject                 DisengageReject,
        locationRequest                 LocationRequest,
        locationConfirm                 LocationConfirm,
        locationReject                  LocationReject,
        infoRequest                     InfoRequest,
        infoRequestResponse             InfoRequestResponse,
        nonStandardMessage              NonStandardMessage,
        unknownMessageResponse          UnknownMessageResponse,
        ...,
        requestInProgress               RequestInProgress,
        resourcesAvailableIndicate      ResourcesAvailableIndicate,
        resourcesAvailableConfirm       ResourcesAvailableConfirm,
        infoRequestAck                  InfoRequestAck,
        infoRequestNak                  InfoRequestNak,
        serviceControlIndication        ServiceControlIndication,
        serviceControlResponse          ServiceControlResponse
}
```

The user should use the test programs in the H.323 tests directory (*h323fw/tests/h323_ras*) as a guide when reading the rest of the procedure.

To encode a RAS message, a variable of the *ASN1T_H225RasMessage* C structure must first be populated with data. This structure is as follows:

```
typedef struct EXTERN ASN1T_H225RasMessage {
   int t;
   union {
      /* t = 1 */
      ASN1T_H225GatekeeperRequest *gatekeeperRequest;
      /* t = 2 */
      ASN1T_H225GatekeeperConfirm *gatekeeperConfirm;
      /* t = 3 */
      ASN1T_H225GatekeeperReject *gatekeeperReject;
      .
      .
      .
      /* t = 33 */
      ASN1TOpenType *extElem1;
   } u;
} ASN1T_H225RasMessage;
```

This structure has two fields: *t* and *u*. The *t* field is defined as follows:

```
#define T_H225RasMessage_gatekeeperRequest 1
#define T_H225RasMessage_gatekeeperConfirm 2
#define T_H225RasMessage_gatekeeperReject 3
.
.
.
#define T_H225RasMessage_extElem1       33
```

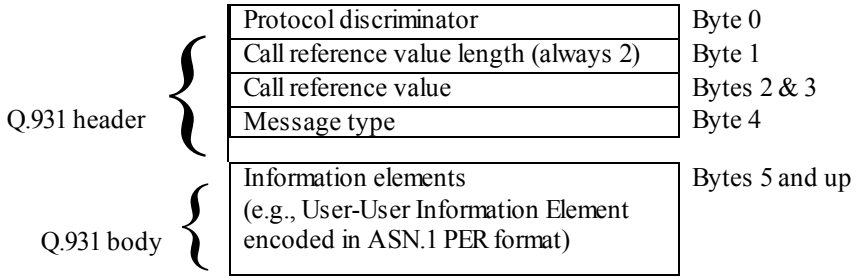
This is a choice of many alternatives: *gatekeeperRequest*, *gatekeeperConfirm*, *gatekeeperReject*… Setting the *t* field member of the generated structure specifies the choice alternative. The value *T_H225RasMessage_gatekeeperRequest* specifies the gatekeeper request option, whereas *T_H225RasMessage_gatekeeperConfirm* specifies the gatekeeper confirm, etc. The data for the structure is set by filling in the union value *u* with data of the selected choice option.

The general procedure to populate the data structure and encode a message is as follows (see *writer.c* for details):

1. Declare an *ASN1CTXT* variable and initialize it using the *rtInitContext* function,
2. Populate the *ASN1T_H225RasMessage* structure with the data to be encoded as described above,
3. Use the *pu_setBuffer* function to specify the buffer the message is to be encoded into,
4. Invoke the *asn1PE_H225RasMessage* function to encode the packet, and
5. Use the *pe_GetMsgPtr* function to get the start address and length of the encoded message component,
6. Use the *pu_freeContext* function to release the *ASN1CTXT* when finished.

Encoding a Q.931 Message

Q.931 messages are another class of H.323 messages. These messages are used for call control between two H.323 terminals. The general format of a Q.931 message includes a *protocol discriminator*, a *call reference value* to distinguish between different calls, a *message type*, and various *information elements* (IE's) as required by the particular message type. The User-User Information Element is used by H.323 to carry the call control information, which is encoded in ASN.1 PER format. The following diagram shows the byte layout for a Q.931 message:



After identifying the call and the message type in the Q.931 header, the IE's must be encoded. The IE's also have a small header comprised of the IE identifier and IE length, after which the IE data follows. For H.323 this is where the PER encoded User-User Information Element goes. Refer to the H.323 and Q.931 specifications for more specific encoding details for the different message types.

The ASN.1 definition for a Q.931 User-User Information Element is as follows:

```
H323-UserInformation ::= SEQUENCE   -- root for all Q.931 related ASN.1
{
      h323-uu-pdu H323-UU-PDU,
      user-data   SEQUENCE
      {
            protocol-discriminator        INTEGER     (0..255),
            user-information        OCTET STRING (SIZE(1..131)),
            ...
      } OPTIONAL,
      ...
}
```

The user should use the test programs in the H.323 tests directory (*h323fw/tests/ h323_q931*) as a guide when reading the rest of the procedure. Additionally, many useful Q.931 utility functions are available in the H.323 src and include directories (*h323fw/src* and *h323fw/include*). These functions are provided for convenience, but not required as part of the ASN H.323 compiled code.

To encode a Q.931 message, a variable of the *ASN1T_H225H323_UserInformation* C structure must first be populated with data. This structure is as follows:

```
typedef struct EXTERN ASN1T_H225H323_UserInformation {
   struct {
      unsigned user_dataPresent : 1;
   } m;
   ASN1T_H225H323_UU_PDU  h323_uu_pdu;
   ASN1T_H225H323_UserInformation_user_data  user_data;
   ASN1TSeqExt  extElem1;

   ASN1T_H225H323_UserInformation () {
      m.user_dataPresent = 0;
   }
} ASN1T_H225H323_UserInformation;
```

This structure has four fields: *m*, *h323_uu_pdu*, *user_data*, and *extElem1*. The *m* field is a bit-field which flags whether the *user_data* field is populated. The data for the structure *h323_uu_pdu* must be set. If applicable, the *user_data* and *extElem1* fields can also be set. The *extElem1* field is a collector for open extension items, as the ASN.1 H323-UserInformation definition is an open sequence.

After the *ASN1T_H225H323_UserInformation* C structure is populated completely, it should be encoded. Once this is completed, the encoded data should be sent as the User-user Information Element within a Q.931 message. The general procedure to do everything is as follows (see *tester.c* for details):

1. Declare an *ASN1CTXT* variable and initialize it using the *rtInitContext* function,

2. Populate the *ASN1T_H225H323_UserInformation* structure with the data to be encoded as described above,

3. Use the *pu_setBuffer* function to specify the buffer the message is to be encoded into,

4. Invoke the *asn1PE_H225H323_UserInformation* function to encode the packet,

5. Use the *pe_GetMsgPtr* function to get the start address and length of the encoded message

component.

6. Use the encoded message component as the User-user Information Element data within the appropriate Q.931 IE, after the appropriate Q.931 header, as described above, and

7. Use the *pu_freeContext* function to release the *ASN1CTXT* when finished.

Encoding a H.245 Message

H.245 messages are the third class of H.323 messages. These messages are used for multimedia control signaling during a call between two H.323 terminals. The ASN.1 definition for a H.245 message is as follows:

```
MultimediaSystemControlMessage      ::=CHOICE
{
      request     RequestMessage,
      response    ResponseMessage,
      command     CommandMessage,
      indication  IndicationMessage,
      ...
}
```

The user should use the test programs in the H.323 tests directory (*h323fw/tests/h323_h245*) as a guide when reading the rest of the procedure.

To encode a H.245 message, a variable of the *ASN1T_H245MultimediaSystemControlMessage* C structure must first be populated with data. This structure is as follows:

```
typedef struct EXTERN ASN1T_H245MultimediaSystemControlMessage {
   int t;
   union {
      /* t = 1 */
      ASN1T_H245RequestMessage *request;
      /* t = 2 */
      ASN1T_H245ResponseMessage *response;
      /* t = 3 */
      ASN1T_H245CommandMessage *command;
      /* t = 4 */
      ASN1T_H245IndicationMessage *indication;
      /* t = 5 */
      ASN1TOpenType *extElem1;
   } u;
} ASN1T_H245MultimediaSystemControlMessage;
```

This structure has two fields: *t* and *u*. The *t* field is defined as follows:

```
#define T_H245MultimediaSystemControlMessage_request 1
#define T_H245MultimediaSystemControlMessage_response 2
#define T_H245MultimediaSystemControlMessage_command 3
#define T_H245MultimediaSystemControlMessage_indication 4
#define T_H245MultimediaSystemControlMessage_extElem1 5
```

This is a choice of four alternatives: *request*, *response*, *command*, or *indication*. Setting the *t* field member of the generated structure specifies the choice alternative. The value

T_H245MultimediaSystemControlMessage_request specifies the request option,
T_H245MultimediaSystemControlMessage_response specifies response,
T_H245MultimediaSystemControlMessage_command specifies command,
T_H245MultimediaSystemControlMessage_indication specifies indication, and
T_H245MultimediaSystemControlMessage_extElem1 specifies extElem1. The data for the structure is set by filling in the union value *u* with data of the selected choice option. The *extElem1* field is a collector for open extension items, as the ASN.1 MultimediaSystemControlMessage definition is an open choice.

The general procedure to do everything is as follows (see *tester.c* for details):

1. Declare an *ASN1CTXT* variable and initialize it using the *rtInitContext* function,
2. Populate the *ASN1T_H245MultimediaSystemControlMessage* structure with the data to be encoded as described above,
3. Use the *pu_setBuffer* function to specify the buffer the message is to be encoded into,
4. Invoke the *asn1PE_H245MultimediaSystemControlMessage* function to encode the packet,
5. Use the *pe_GetMsgPtr* function to get the start address and length of the encoded message component, and
6. Use the *pu_freeContext* function to release the *ASN1CTXT* when finished.

Decoding H.323 Packet Structures

This section describes the procedure to decode different types of H.323 messages. This is the inverse of the encoding procedures presented earlier.

Decoding a RAS Message

The user should use the test programs in the H.323 tests directory (*h323fw/tests/h323_ras*) as a guide when reading the rest of the procedure.

The procedure to decode the RAS message structure is as follows:

1. Declare an *ASN1CTXT* variable and initialize it using the *rtInitContext* function,
2. Use the *pu_setBuffer* function to set the pointer and length of the buffer containing the RAS message structure to be decoded,
3. Invoke the *asn1PD_H225RasMessage* function to decode the data,
4. Access the *ASN1T_H225RasMessage* structure to get at the decoded data fields,
5. Use the *pu_freeContext* function to release the *ASN1CTXT* when finished.

The C structures that are used to describe a RAS message were shown in the section on encoding.

Decoding a Q.931 Message

The user should use the test programs in the H.323 tests directory (*h323fw/tests/h323_q931*) as a guide when reading the rest of the procedure. Additionally, many useful Q.931 utility functions are available in the H.323 src and include directories (*h323fw/src* and *h323fw/include*). These functions are provided for convenience, but not required as part of the H.323 ASN compiled code.

The procedure to decode the Q.931 message structure is as follows:

1. Declare a *Q931Message* variable and initialize it using the *memset* function,
2. Declare an *ASN1CTXT* variable and initialize it using the *rtInitContext* function,
3. Set the *Q931Message.ctxt_p* pointer to point to the *ASN1CTXT* variable (if you wish to keep the context pointer associated with the Q.931 message),
4. Use the *q931_Decode* function to decode the Q.931 message from a data buffer into the *Q931Message* variable,
5. Use the *q931_GetIE* function with the *Q931UserUserIE* code to extract the User-user Information Element data buffer from the Q.931 message,
6. Use the *pu_setBuffer* function to set the pointer and length of the buffer containing the Q.931 Information Element message structure to be decoded,
7. Invoke the *asn1PD_H225H323_UserInformation* function to decode the data,
8. Access the *ASN1T_H225H323_UserInformation* structure to get at the decoded data fields,
9. Use the *pu_freeContext* function to release the *ASN1CTXT* when finished.

The C structures that are used to describe a Q.931 message were shown in the section on encoding.

Decoding a H.245 Message

The user should use the test programs in the H.323 tests directory (*h323fw/tests/h323_h245*) as a guide when reading the rest of the procedure.

The procedure to decode the H.245 message structure is as follows:

1. Declare an *ASN1CTXT* variable and initialize it using the *rtInitContext* function,
2. Use the *pu_setBuffer* function to set the pointer and length of the buffer containing the H.245 message structure to be decoded,
3. Invoke the *asn1PD_H245MultimediaSystemControlMessage* function to decode the data,
4. Access the *ASN1T_H245MultimediaSystemControlMessage* structure to get at the decoded data fields,
5. Use the *pu_freeContext* function to release the *ASN1CTXT* when finished.

The C structures that are used to describe a H.245 message were shown in the section on encoding.