H.323 Framework for C
PER Encode/Decode API
User's Guide

Objective Systems, Inc. April 2005

Introduction

The H.323 Framework for C PER Encode/Decode API is a package of functions for encoding and decoding messages from the H.323 ASN.1 specifications using the Packed Encoding Rules (PER) as defined in ITU standard X.691. 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.

This API has been developed in the C programming language. The Objective Systems ASN1C compiler was used to generate the initial structures and encode/decode functions. 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).

Installation

The evaluation version of this package is an add-on to an existing ASN1C compiler installation. It is therefore necessary to have ASN1C installed before installing this package. An ASN1C evaluation version may be obtained from http://www.obj-sys.com.

The steps to install this package are as follows (note: in all of these steps, the ASN1C version number is assumed to be 573. If you are using a different version of ASN1C, substitute the version number you are using).

1. Unzip this package in the 'c' subdirectory of the ASN1C installation. For example, if ASN1C is installed in the 'c:\acv573' directory, unzip this package in the 'c:\acv573\c' subdirectory (note: for Linux or UNIX, the base directory name would be of the form ‘asn1c-v573’, in this case the package should be unpacked into the ‘asn1c-v573/c’ subdirectory).

2. Open a terminal window. (On Windows, go to ‘Start -> Accessories -> Command Prompt’ to get a terminal window).

3. Change directory to the 'h323fw/src’ subdirectory:

   On Windows: `cd c:\acv573\c\h323fw\src`

   On Linux/UNIX : `cd asn1c-v573/c/h323fw/src`

4. Execute the make utility to compile the specifications.

   On Windows, the Visual Studio nmake tool can be used.

   (note: nmake is a make utility program that comes with the Microsoft Visual C++ compiler. It may be necessary to execute the batch file vcvars32.bat that comes with Visual C++ in order to set up the environment variables to use this utility).

   On Linux/UNIX, the standard make utility can be used.

5. The H.323 specifications should now be compiled and the results stored in the h323.lib library in the lib subdirectory. Note that this is a single-threaded static library. Other library configurations (static multi-threaded or DLL) will require modifications to the build procedure.

6. You can test out linking and execution by executing the sample programs in the sample subdirectory.

Contents of the Package

The following diagram shows the directory tree structure that comprises the H.323 PER encode/decode package:

```
h323fw
 |
 +- doc
 |
 +- lib
 |
 +- src
 |
 +- specs
 |
 +- sample
    |
    +- h323_ras
    |
    +- h323_ui
```

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 contains the resulting H.323 run-time library after the generated code is compiled. Applications would need to be linked with this run-time library as well as the base run-time libraries contained within the ASN1C package.
- src – this directory contains all of the ASN1C compiler generated C source files
- specs – this directory contains the H.323 ASN.1 specifications
- sample – this directory contains sample programs that show how to use the API.

Getting Started

Once the library is installed, the encode/decode functionality can be tested by running the sample programs. Each sample program contains a reader and a writer program. The sample is built by executing the makefile in the sample subdirectory. The writer program can then be executed to populate a test data structure, encode the data, and then writer the corresponding encoded binary data to a disk file. The reader program can then be executed to read the file and decode the record.

The following sections provide greater detail on the procedure of encoding and decoding H.323 packets.

Encoding H.323 Packet Structures

This section describes to procedure to encode different types of H.323 messages.

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 *rtFreeContext* 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:

Protocol discriminator	Byte 0
Call reference value length (always 2)	Byte 1

Q.931 header	Protocol discriminator	Byte 0
	Call reference value length (always 2)	Byte 1
	Call reference value	Bytes 2 & 3
	Message type	Byte 4
Q.931 body	Information elements (e.g., User-User Information Element encoded in ASN.1 PER format)	Bytes 5 and up

After identifying the call and the message type in the Q.931 header, the IE's must be encoded. The IE's 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 sample programs in the H.323 sample directory (*h323fw/sample/ h323_ui*) as a guide when reading the rest of the procedure.

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 procedure to do this is as follows:

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 *rtFreeContext* 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,
      ...
}
```

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 *rtFreeContext* 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 procedure to decode a RAS message 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 *rtFreeContext* 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 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 *rtFreeContext* 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 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 *rtFreeContext* 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.