VistA Messaging Services

Health Level Seven Optimized (HLO) Developer Manual

Software HL*1.6
Document Revision 1.2

Department of Veterans Affairs
VA Office of Information and Technology (OI&T)
Veterans Health Information Technology (VHIT)
VistA Messaging Services

(*This page left blank for two sided printing*)

Revision History

Date	Revision	Description	Author
5/20/09	1	Created document (VMS_CR1261)	Jim Moore
8/4/09	1.1	Technical Edit (VMS_CR1261)	Brian Lynch
9/28/09	1.2	Technical Edit (VMS_CR1261)	Brian Lynch

(*This page left blank for two sided printing*)

Contents

1.0 INTRODUCTION 1

1.1 SCOPE OF THIS MANUAL 1

1.2 THE HL7 STANDARD 2

1.2.1 HL7 Version 2

1.2.2 Supported Communication Protocols 2

1.2.3 HL7 Messages 2

1.2.3.1 Individual Messages 2

1.2.3.2 Batch Messages 4

1.2.4 Local Customizations 4

1.2.5 Data Types and Tables 4

1.2.6 Message Acknowledgments 5

1.2.7 Message Delimiters 5

1.2.8 Escape Sequences 5

1.3 APPLICATION'S RESPONSIBILITIES VERSUS HLO'S RESPONSIBILITIES 6

1.3.1 HLO 6

1.3.2 Application 6

1.4 HLO CLIENT-SERVER BEHAVIOR 7

1.4.1 HLO Client Behavior 7

1.4.1.1 The Outgoing Queue Structure 7

1.4.1.2 The Client Algorithm 7

1.4.2 HLO Server Behavior 8

1.4.2.1 The Server Algorithm 8

1.4.2.2 The Incoming Queue Structure 9

1.5 FEATURES NOT SUPPORTED BY HLO 9

1.6 SPECIAL CONSIDERATIONS FOR INTERFACING HLO TO COTS, MIDDLEWARE, OR OTHER MESSAGING APPLICATIONS 9

2.0 BUILDING AND SENDING HL7 MESSAGES 10

2.1 OVERVIEW 10

2.2 CREATE AN ENTRY IN THE HLO APPLICATION REGISTRY FILE FOR THE SENDING APPLICATION 12

2.3 CREATE ENTRIES IN THE HL LOGICAL LINK FILE FOR EACH DESTINATION 13

2.4 THE MSH, BHS, AND BTS SEGMENTS 14

2.4.1 HLO's MSH Segment 14

2.4.2 HLO's BHS Segment 15

2.4.3 HLO's BTS Segment 16

2.5 LIST OF MESSAGE BUILDING APIs....17
2.6 THE HLO FRAMEWORK FOR MESSAGE BUILDING....18
2.6.1 New-Style Message Building....18
2.6.1.1 Pattern 1 – Building a Segment....18
2.6.1.2 Pattern 2 – Building an Individual Message....19
2.6.1.3 Pattern 3 – Building a Batch Message....19
2.6.2 Traditional Style Message Building....20
2.6.2.1 Pattern 4 Building an Individual Message in the Traditional Style....20
2.6.3 Hybrid Style Message Building....20
2.6.3.1 Pattern 5 – Mixing Traditional Style and New Style Segment Builders....20
2.6.3.2 Pattern 6 – Traditional Message Building without Protocols....21
2.7 APPLICATION CALLBACKS....21
2.7.1 Callback for Application Acknowledgments....21
2.7.2 Callback for Commit Acknowledgments....21
2.7.3 Callbacks for Messages that Fail to Transmit....22
2.8 SEQUENCE QUEUES....22
2.8.1 The Algorithm....22
2.8.1.1 Client Role:....22
2.8.1.2 Server Role:....23
2.8.2 Using Sequence Queues....23
2.9 HLO SUBSCRIPTION REGISTRY....23
2.9.1 The User Interface for Creating Subscription Lists....24
2.9.2 The Programming Interface for Creating Subscription Lists....25
2.10 TURNING MESSAGING ON/OFF....26
3.0 RECEIVING MESSAGES....27
3.1 SETTING UP THE RECEIVING APPLICATION IN THE HLO APPLICATION REGISTRY....27
3.2 LIST OF MESSAGE PARSING APIs....29
3.3 HLO PARSES THE MESSAGE HEADER....30
3.3.1 Parsing the MSH Segment....30
3.3.2 Parsing the BHS Segment....31
3.4 STEPPING THROUGH AN INDIVIDUAL MESSAGE....32
3.4.1 Method 1 – Using $$NEXTSEG^HLOPRS....32
3.4.2 Method 2 – Using $$HLNEXT^HLOMSG....33
3.4.3 Examples of Message Parsing....33
3.5 STEPPING THROUGH A BATCH OF MESSAGES....34
4.0 ACKNOWLEDGEMENT MESSAGES....36

4.1 REQUESTING ACKNOWLEDGMENTS 36
4.2 RETURNING ACKNOWLEDGMENTS 38
4.2.1 Commit Acknowledgments 38
4.2.2 Application Acknowledgments....38
4.2.3 Determining the Return Destination 40
5.0 API REFERENCE GUIDE 41
5.1 SENDING MESSAGES 42
5.1.1 Start Building a New Message....42
5.1.2 Start Building a New Batch of Messages....43
5.1.3 Add a New Message to a Batch 44
5.1.4 Add a Segment to a Message 45
5.1.5 Move Traditional-Style Message Array into an HLO Message....45
5.1.6 Move Traditional-Style Segment Array into HLO 46
5.1.7 Setting Data Types into a Segment....47
5.1.7.1 Set Value (Unspecified Data Type).... 47
5.1.7.2 Set Address.... 48
5.1.7.3 Set Coded Element 49
5.1.7.4 Set Coded Element with No Exceptions.... 50
5.1.7.5 Set Coded Value with Exceptions 51
5.1.7.6 Set Date 51
5.1.7.7 Set Hierarchic Designator.... 52
5.1.7.8 Set Timestamp.... 53
5.1.7.9 Set Extended Person Name 53
5.1.8 Completing and Sending Messages 54
5.1.8.1 Send a Single Message 55
5.1.8.2 Send Messages to a List of Destinations 56
5.1.8.3 Send Messages via the HLO Subscription Registry to a List of Subscribers.... 57
5.1.8.4 Send Messages via HL7 1.6 Protocol Setup 58
5.2 PARSING MESSAGES 59
5.2.1 Start Parsing a Message 59
5.2.2 Advance to the Next Message within a Batch 61
5.2.3 Advance to and Parse the Next Segment 61
5.2.4 Get Next Segment....62
5.2.5 Get Message ID 63
5.2.6 Parsing Data Types 63
5.2.6.1 Get Value (Unspecified Data Type) 63

5.2.6.2 Get Address ... 64
5.2.6.3 Get Coded Element ... 65
5.2.6.4 Get Coded Element with No Exceptions ... 65
5.2.6.5 Get Coded Element with Exceptions ... 66
5.2.6.6 Get Date ... 67
5.2.6.7 Get Hierarchic Designator ... 68
5.2.6.8 Get Timestamp ... 68
5.2.6.9 Get Extended Person Name ... 69
5.3 RETURNING APPLICATION ACKNOWLEDGMENTS ... 70
5.3.1 Begin Application Acknowledgment for an Individual Message ... 70
5.3.2 Begin Batch Application Acknowledgement ... 71
5.3.3 Add an Application Acknowledgement to a Batch ... 72
5.3.4 Send the Application Acknowledgement. ... 73
5.4 SUBSCRIPTION REGISTRY ... 73
5.4.1 Create an Entry in the Subscription Registry ... 73
5.4.2 Add a New Recipient to a Subscription Registry Entry ... 74
5.4.3 Terminate a Recipient from a Subscription Registry Entry ... 75
5.4.4 Check Subscription Registry Entry for a Recipient ... 76
5.4.5 Retrieve Next Recipient from a Subscription Registry Entry ... 76
5.4.6 Build a Subscription Registry Index ... 77
5.4.7 Find a Subscription Registry Entry ... 77
5.5 MISCELLANEOUS APIS ... 78
5.5.1 Resend a Message ... 78
5.5.2 Reprocess a Message ... 78
5.5.3 Reprocess a Message Immediately ... 78
5.5.4 Reset the Purge Date and Time for a Message ... 79
5.5.5 Count Messages on All Queues ... 80
5.5.6 Count Messages on Outgoing Queues ... 80
5.5.7 Count of Messages on Incoming Queues ... 81
5.5.8 Count of the number of messages on the sequence queues. ... 81
6.0 CODE EXAMPLES ... 82
6.1 HLODEM1 ... 82
6.2 HLODEM2 ... 86
6.3 HLODEM3 ... 90
6.4 HLODEM4 ... 92
6.5 HLODEM5 ... 93

6.6 HLODEM6 96
6.7 HLODEM7 99
6.8 HLODEM8 100
6.9 HLODEM9 101
6.10 HLODEM10 102
7.0 QUICK OVERVIEW OF THE HLO USER INTERFACE 105
7.1 HLO MAIN MENU 105
7.2 HLO SYSTEM MONITOR 106
8.0 TROUBLESHOOTING FOR THE DEVELOPER 108
8.1 THE CLIENT TRACE TOOL 108
8.2 THE SERVER TRACE TOOL 110

(*This page left blank for two sided printing*)

1.0 Introduction

VistA's Health Level Seven Optimized (HLO) software is VistA's newest software that supports Health Level Seven (HL7) messaging for VistA MUMPS (M) applications. It was released in patch HL*1.6*126. It consists of:

1. A set of Application Programming Interfaces (APIs) that VistA M developers use to build and transmit HL7 messages, and receive and parse HL7 messages.
2. A messaging engine, consisting of a client and a server, that sends and receives HL7 messages between systems.
3. A user interface that sites use to control and monitor the HL7 messaging, and view messages, message errors, and message statistics.

This manual is a guide to those who are developing HL7 interfaces using HLO, either between two VistA systems, or between a VistA system and a Commercial Off-the-Shelf (COTS) or other non-VistA system. It assumes familiarity with the HL7 standard.

HLO coexists with the earlier HL7 software. *In this document the older software will be referred to as'HL7 1.6'*. Messaging applications that were developed prior to HLO will continue to use HL7 1.6, though it is recommended that new applications use HLO.

Note: In this document the older software will be referred to as'HL7 1.6'.

1.1 Scope of this Manual

This manual is concerned with the technical aspects of using HLO to build a messaging application. Among the steps involved in interfacing systems that are **not** discussed are:

- Analysis of the data that needs to be exchanged. This requires individuals with in-depth knowledge of the information domain and the functions of the systems being interfaced.
- Determining the HL7 messages that are needed to exchange the information.
- Detailed analysis of the data and compatibility issues between the two systems. For example, if the two systems use different code systems to record lab tests, then conversion of the data from one code system to another may be a factor to consider.
- Negotiation, documentation, and approval of the interface specification.
- Determination of the trigger events that must be responded to by the messaging. For example, in a patient appointment scheduling system, is a message needed when an appointment is cancelled? The HL7 standard provides a wide variety of event types that may be implemented.
- Detailed analysis of the sending application software to determine how to insert hooks to implement the desired trigger events. For example, if an interface is to include notification of an appointment cancellation then that requires a detailed analysis of the scheduling system software to determine at which point, or points, the hooks need to be inserted. It may be within a routine, an option, protocol, or other components that make up software within VistA. When those points are determined, the interface developer needs to insert a call to a routine that uses the HLO APIs to build and send the appropriate HL7 message – that IS a topic within the scope of this manual.

1.2 The HL7 Standard

HL7 is an application protocol for the electronic exchange of data between information systems in the healthcare environment. It facilitates the exchange of data by specifying a standard set of messages, including the content and format of the messages and the encoding of the data fields within the message. The HL7 standard provides flexibility by allowing the standard set of messages to be extended for specific needs, subject to negotiation between the systems being interfaced to one another.

HL7 does not specify the particular communication protocol or technology to use for the exchange of data.

1.2.1 HL7 Version

HLO was designed to support version 2.4 of the HL7 standard, using the traditional delimited format for HL7 messages. The application may designate other versions of the standard in the message header, but that won't otherwise affect how HLO formats the message.

1.2.2 Supported Communication Protocols

HLO currently supports messaging only via TCP, using the set of APIs provided by Cache to read and write data over TCP communication channels. HLO implements the HL7 Standard's Minimal Lower Layer Protocol (MLLP) to package messages for transmission via TCP.

1.2.3 HL7 Messages

HLO implements both individual messages and batch messages.

1.2.3.1 Individual Messages

An individual HL7 message is the atomic unit for transferring data between systems in the HL7 standard. The HL7 standard defines numerous types of messages. The type of message is identified by the *message type* and *event type*, each of which is a three-letter code. For example, for the ADT~A01 message, the message type is ADT (admission, discharge, transfer) and the event type is A01 (admission/visit notification). The HL7 standard defines the content and format of each type of message. Since the HL7 standard is evolving, there are different versions of the messages.

A message is composed of groups of segments. A *group* is an ordered set of related segments.

A *segment* is an ordered set of related fields. The first three characters of a segment is a code that identifies the *segment type*. For example, the PID segment is the patient identification segment and contains information that identifies the patient.

The first segment in a message is always the MSH segment, the *message header*. It contains information about the message, such as the type of message, the version, the sending application, the receiving application, and the date/time of the message. It also contains the *Message Control ID*, which is used to uniquely identify each message.

A *field* is an item of information, such as a person's name or address or weight. Fields may repeat within a segment. For example, the NAME field within the PID segment may be repeating since a person may have more than one name. On the other hand, the SEX field is not allowed to repeat.

A field can have multiple parts called *components*. For example, the Patient Address field of the PID segment contains the street, state, city, zip code, etc., which are components of the field. In turn, components can have multiple parts called *subcomponents*.

Special characters called *message delimiters* are used to mark the boundaries between the segments, fields, components, and subcomponents. They are:

- The segment terminator marks the boundaries between segments. The segment terminator is always the carriage return character.
- The field separator marks the boundary between fields.
- The component separator marks the boundary between the components of a field.
- The repetition separator marks the boundary between repetitions of a field.
- The escape character marks the start and end of escape sequences within a field. An escape sequence is an instruction for special handling of the data.
- The subcomponent separator marks the boundary between the subcomponents within a subcomponent.

Here is an example of an HL7 message:

```
MSH|^~\&|ORDER|BOSTON VAMC|RESULTS|500|19900314130405||ORU^R01|523123|D|2.3|
PID||7777790^2^M10||HL7Patient^One|||||||123456789|
OBR||2930423.08^1^L||199304230800||||||DERMATOLOGY|
OBX|CE|10040|OV|1^0^0^0^0|
OBX|CE|11041|PR|
OBX|CE|216.6|P|
OBX|ST|VW^WEIGHT^L||120|KG
OBX|ST|VB^BLOOD PRESSURE^L||120/80|MM HG
OBX|ST|VT^TEMPERATURE^L||99|C
OBX|ST|VP^PULSE^L||75|/MIN
```

The following is a partial analysis of the message:

- The first line of the message is the message header (MSH) segment. Within the MSH segment, we see that:
  - The field separator is ‘|’, the component separator is ‘^’, the repetition separator is ‘~’, the escape character is ‘\’ and the subcomponent separator is ‘&’. This is the recommended set.
  - The sending application is ORDER and the receiving application is RESULTS.
  - The sending facility is BOSTON VAMC and the receiving facility is 500. (HLO uses the domain name and station number to identify the facilities. The standard often leaves such choices to the applications to negotiate.)
  - The message type is Observation Result/Unsolicited (ORU) and the event type is an unsolicited transmission of an observation message (R01).
- The second line of the message is the second segment, Patient Identification (PID).
- The third line of the message is the third segment, an Observation Request (OBR).
- The subsequent lines of the message are multiple Observation/Results (OBX) segments.

The HL7 standard provides a variety of message types and event types relevant to many areas in health care, including:

- Patient Administration
- Order Entry
- General Queries
- Financial Management
- Observation Reporting
- Master Files
- Medical Records/Information
- Scheduling
- Patient Referral
- Patient Care

Note on Notation: The nth field in a segment will be referenced by the three- letter code and the field number as follows:'MSH-1' is the first field in the MSH segment.

1.2.3.2 Batch Messages

A batch message is a message that contains individual messages. The batch message itself doesn't convey much information; rather, it is the individual messages within the batch that contain the information.

A batch message starts with the *batch header segment* (BHS**)**. Instead of containing a Message Control ID as in the MSH segment, it contains the *Batch Control ID* that serves the same purpose of uniquely identifying the message. The batch message ends with the *batch trailer segment* (BTS**)**.

The individual messages within the batch are exactly the same as when they don't appear within a batch. A batch message may contain different types of messages, though generally batch messages contain messages of all the same type.

1.2.4 Local Customizations

The standard also permits organizational-specific customizations to messages through a number of mechanisms including:

- Locally defined message types. Their three-letter code must begin with a'Z'.
- Locally defined event types. Their three-letter code must begin with a'Z'.
- Locally defined segment types. Their three-letter code must begin with a'Z'.
- Optional fields, components, subcomponents, and repeating fields.
- User-defined tables.

1.2.5 Data Types and Tables

One of the issues involved in exchanging data between disparate systems is the format of the data and units of measurement. The HL7 standard provides a wide variety of data types to specify data as divergent as address, time, lab results, lab test results, etc. HLO supports some of the most common of the specialized data types, but not all of them. The support provided is in the form of specialized APIs for setting data types into a message under construction, or parsing the data types from a segment while reading the message.

Many of the data types reference tables, both tables defined within the HL7 standard as well as external tables and user-defined tables. The HLO software does not keep a database of those tables. It is the application's responsibility to insure that the codes it uses within its messages conform to the tables. An exception pertains to the codes used within the message header, for example, the acknowledgment type codes, for which HLO is responsible.

1.2.6 Message Acknowledgments

When an application initiates the exchange of messages, the return of a message called an *acknowledgement message* from the receiving system may be required as part of the defined transaction. The HL7 standard defines two different acknowledgment protocols, *original mode* and *enhanced mode*. HLO supports only enhanced mode acknowledgments, which have two-phases, both of which are optional:

- An accept acknowledgement (also called a commit acknowledgement) confirms that the receiving system has received the message and has committed it to safe storage.
- An application acknowledgement confirms that the receiving application processed the sender's message and may include additional information, for example, the reply to a query message.

The type of acknowledgement returned for any given message depends on the negotiated interface between the sending and receiving applications.

Note: Commit acknowledgments are recommended to guarantee message delivery.

1.2.7 Message Delimiters

The choice of characters to use as the message delimiters is left to the application. However, HL7 recommends this set:

Field Separator:	|
Component Separator:	^
Repetition Separator:	~
Escape Character:	\
Subcomponent Separator:	&

HLO allows the application to specify the set of delimiters to use, but defaults to the recommended set. When HLO generates accept acknowledgments, it always uses the recommended set of delimiters. When an application calls the HLO APIs to generate an application acknowledgment, the application must specify the delimiters to use if it wants other than the recommended set – i.e., HLO does not default to the same set as the original message.

1.2.8 Escape Sequences

Suppose a data value contains one of the message delimiters, for example, the component separator is "^" and a data value is "a^39". If the value were simply inserted into a field, *it would be impossible to correctly parse it back out from the message* because the parsing application would instead interpret it as two components, the first with value "a" and the second with value "39". To provide for this the HL7 standard requires that message delimiters that appear within a value be replaced by the escape sequence for that delimiter. When parsing the value out of the message, the escape sequence must then be replaced by the original character.

The escape sequences for the message delimiters are:

\F\	field separator
\S\	component separator
\T\	subcomponent separator
\R\	repetition separator
\E\	escape character

In the example above, to set the field value of “a^39” into the message, the “^” would be replaced with the escape sequence “\S\”, and instead of “a^39” the value would appear within the segment as “a\S\39”.

HLO supports these escape sequences, fully automating the process provided that the application uses the message building and parsing APIs.

The standard also provides these escape sequences which HLO does NOT support:

\H\	start highlighting
\N\	normal text (end highlighting)
\Xdddd…\	hexadecimal data
\Zdddd…\	locally defined escape sequence

1.3 Application’s Responsibilities Versus HLO’s Responsibilities

1.3.1 HLO

- HLO is responsible for message delivery. It accepts messages from the sending application for transmission to remote destinations. It receives messages from remote destinations and passes them to the receiving application for processing.
- HLO provides the application with tools for message construction and parsing.
- HLO builds the message header for the sending application.
- HLO parses the message header for the receiving application.
- When an application builds an HL7 message with the HLO APIs, HLO will automatically replace message delimiters that occur within the data with the appropriate escape sequence. When an application parses a message with the HLO APIs, HLO will automatically replace the escape sequence with the original character.
- HLO purges completed messages after a time period specified by the system administrator.
- HLO handles the exchange of commit acknowledgments if requested by the sending application.

1.3.2 Application

- The application is responsible for the content of the messages, for building and parsing the content of its messages, and for processing them.
- HLO is only responsible for building the MSH and BHS header segments. The application is responsible for the building of the other segments, and for insuring that its segments conform to the HL7 standard.
- The application may choose to hard-code its message building or parsing rather than using the HLO APIs. However, in that case, the application is responsible for handling escape sequences.
- It is the application’s responsibility to determine whether or not to utilize commit acknowledgments and/or application acknowledgments. Commit acknowledgments are recommended to guarantee message delivery.
- The application is responsible for insuring that when it references tables that it uses valid codes. HLO does not maintain a database of tables referenced by HL7 messages.

1.4 HLO Client-Server Behavior

When a messaging transaction occurs between two systems, one system always acts as the server and the other system acts as the client. The client initiates the transaction. The server passively monitors the communication channel, waiting for a remote client to initiate a message exchange.

1.4.1 HLO Client Behavior

VistA applications generate messages and place them on outgoing queues. Each queue contains messages meant for a particular destination. There are always HLO client processes running in the background, looking for queued outgoing messages. The client process's job is to transmit those messages to a server at the remote destination.

1.4.1.1 The Outgoing Queue Structure

The outgoing queue is a special cross-reference on the HLO MESSAGES file (#778) with the following format. It has one more level than the queues of HL7 1.6 because it allows multiple queues to be associated with a specific HL Logical Link.

^HLB("QUEUE","OUT",<name of HL Logical Link>:<destination port number>,<*queue name>,<list of messages IENS, file #778, pending on this queue>)

*Sending applications may designate their own private queues. The name should be name-spaced. There is a default queue named "DEFAULT' for messages not assigned to a private queue.

1.4.1.2 The Client Algorithm

For each outgoing queue that has pending messages:

- The client locks the queue to insure that it has exclusive access.
- The client attempts to establish a connection to the remote server. The client will timeout after 30 seconds if the server does not accept the connection request, in which case the client will put this link on the list of down links, unlock the queue, and look for another queue with pending messages. A down queue is ignored for 30 seconds before another client process will again attempt to process it.
- For each message on the queue, up to a limit of 1,000 messages at a time for a particular queue, the client process:
  - Reads the message from the HLO files where it is stored.
  - Writes the message over the open communications channel to the remote HL7 server.
  - Updates the message status, removes the message from the queue, and moves on to sending the next message on the queue if a commit acknowledgment is not required.
  - Attempts to read from the open communications channel if a commit acknowledgment is required.
    - ❖ The client will timeout after 20+ seconds if the remote server fails to return the requested commit acknowledgment. If a timeout does occur, the client will close the connection, and will attempt once to re-open the connection and send the message again, and again try to read the return commit acknowledgment. If that fails, the client will mark the link as down for 30 seconds and move on to the next queue with pending messages. That queue will NOT advance until the message is transmitted and a commit ack is returned.

    - ❖ If the commit acknowledgment is received, the client will update the status of the message and remove the message from the queue. If the sending application requested notification of the commit acknowledgment via a callback (section 2.7.2) the message is placed on an incoming queue to be passed to the application.
- The closes the connection, unlocks the queue, and moves on to the next queue that has messages pending transmission when the client finishes processing a queue, or reaches the limit of 1,000 messages.

1.4.2 HLO Server Behavior

The server's job is to monitor a communications channel, waiting for a remote client to initiate the exchange of messages.

1.4.2.1 The Server Algorithm

- The server monitors the communications channel until it receives a connection request from a client. When it receives a connection request, it spawns another process to handle the messaging with that client and then continues to monitor for more connection requests. This means the server is multi-threaded, allowing it to handle multiple concurrent clients. The following steps apply to the newly spawned child process.
- The server reads messages over the open communication channel until the client terminates the session, at which point this server process terminates. For each message:
  - o The server reads from the communication channel looking for the delimiter marking the beginning of an HL7 message. It continues to read until either it reaches the delimiter marking the end of the message or a read timeout occurs after 20 seconds. If a read timeout occurs, the child server process terminates.
  - o The server parses the individual fields from the message header.
  - o Based on the message header, the server will determine an error if:
    - ❖ There is no Message Control ID specified.
    - ❖ The message is an application acknowledgment and the original message is not found in the HLO MESSAGES FILE (#778).
    - ❖ The message is an application acknowledgment and the original message was already acknowledged.
    - ❖ The Receiving Facility in the message header does not match the server's system.
    - ❖ The Processing ID in the message header doesn't match the system.
    - ❖ The Receiving Application is not found in the HLO APPLICATION REGISTRY file (#779.2) file. The HLO Application Registry is used to lookup what application routine should be executed to process the message.
  - o If the server determines an error, and a commit acknowledgement was requested, a commit acknowledgement is returned with the CE code in the MSA segment. The message status is set to ERROR and the message is not passed to the application.
  - o Also, based on the message header, the server determines if the message is a duplicate.
    - ❖ Messages are frequently received multiple times due to communication failures. For example, if communication fails after the client sends a message, but before the client receives a return commit acknowledgement, then the client would generally resend the message. So the HLO server does a lookup based on the Sending Facility, Sending Application, and Message ID to determine if the message was previously received. If a match is found, the server determines that the message was previously received, and so it is not processed a second time. If a commit acknowledgement was requested, the server will return a commit acknowledgement based on the commit acknowledgement that was returned for the original message.
- If the message is not a duplicate nor an error:
  - o The server returns a commit acknowledgement if one was requested. The MSA segment will have the CA code.

- The server sets the message onto the incoming queue. The server determines the application routine by doing a lookup on the HLO APPLICATION REGISTRY file (#779.2). Another HLO process is responsible for taking the messages off the incoming queue one-by-one and passing them to the application routine to process.

1.4.2.2 The Incoming Queue Structure

The incoming queue is a special cross-reference on the HLO MESSAGES file (#778) with the following format. It has one more level than the queues of HL7 1.6, because it allows multiple queues to be associated with a sending facility.

^HLB("QUEUE","IN",<HL Logical Link>ORSending Facility>,<**queue name*>,<list of messages IENS, file #778, pending on this queue>)

*Receiving applications may designate their own private queues. The name should be name-spaced. There is a default queue named "DEFAULT' for messages not assigned to a private queue.

1.5 Features Not Supported by HLO

1. Original Mode Acknowledgments
2. Synchronous (real-time) communication between the sending and receiving application.
3. Sequence Number Protocol
4. Transfer of messages via files using the FHS and FTS segments.

1.6 Special Considerations for Interfacing HLO to COTS, Middleware, or Other Messaging Applications

1. The Message Control ID and Batch Control ID must be unique.

The Control ID that appears in the message header must be unique within the domain of {sending facility, sending application}. HLO insures that its Control ID's are unique by incorporating the station number. A common mistake is to base the Control ID on a timestamp, which may not insure a unique Control ID.

2. Acknowledgment types are not interchangeable

HLO will not accept an application acknowledgement in lieu of a commit acknowledgement. For example, if the remote server returns the AA code in the MSA segment instead of the CA code, the status of the message will be set to error.

3. Commit acknowledgments are synchronous.

The HLO client expects that the commit acknowledgement be returned synchronously over the same open connection as the original message, and the HLO Server expects to return the commit acknowledgment in the same manner. A commit acknowledgment returned asynchronously over another connection will not be accepted.

4. Batch messaging is not standard.

Batch message processing under HLO incorporates several non-standard features that may prevent batch messaging with COTS systems without customization. These features include:

a. The standard does not provide for a Processing ID in the batch header (BHS segment) as it does in the MSH segment. HLO requires that the Processing ID be in the Batch Name/ID/Type field, sequence 9, of the BHS segment.

b. The standard does not provide for a commit ack to be requested via the batch header. HLO also uses the Batch Name/ID/Type field, sequence 9, of the BHS segment for that purpose.

c. The HL7 standard provides several different methods for returning application acknowledgments in response to a batch of messages. For example, the standard allows the application to return an acknowledgment for every individual message, or to return acknowledgments for only the messages that encountered an error. HLO leaves that to the application to decide, but if the sending application requests acknowledgments for a batch of messages then the receiving application must return the acknowledgments within a single batch of messages and not in multiple batches or as individual messages. The Reference Batch Control ID field (Sequence 12) is used to cross-reference a batch of application acknowledgments to the original batch of messages.

5. Guaranteeing the sequence of message delivery.

When messaging occurs between two applications using HLO with no middleware involved, HLO will insure that messages are delivered to the receiving application in the same order that they were generated by the sending application. However, when other configurations are employed, such as the use of an interface engine, the sequence of the messages cannot be guaranteed by HLO without special handling. HLO provides Sequence Queues that applications may use to guarantee the order of their messages. Sequence Queues use application acknowledgments to insure that a message placed on a Sequence Queue is not delivered until the preceding message is received by the remote Receiving Application. The throughput of a single Sequence Queue, because of its reliance on the exchange of application acknowledgments, is quite slow, so Sequence Queues may not be appropriate for high-volume messaging applications.

2.0 Building and Sending HL7 Messages

2.1 Overview

HLO provides the developer with a set of APIs for building messaging applications. The following is an outline of the development process:

Step 0: HLO needs to be installed, configured, and running on the sending system. An HL7 server, either HLO or other, needs to be running on the receiving system, with TCP connectivity between the two systems.

Step 1: Create the necessary table entries:

- An entry is needed in the HL LOGICAL LINK file (#870) file for each destination.
- An entry is needed in the HLO APPLICATION REGISTRY file (#779.2) for the sending application.
- HLO does not require the use of protocols, but does support their use. If used, entries are needed for the event and subscriber protocols in the PROTOCOL file (#101), and the HL7 APPLICATION PARAMETER file (#771). Instructions for that setup can be found in the *VistA Health Level Seven (HL7) Site Manager & Developer Manual, Version 1.6*56.*

<u>Step 2:</u> Using the HLO APIs, write a routine to build and send the message.

- The developer needs to build the body of the HL7 message segment-by-segment, excluding message headers. The developer is responsible for insuring the content and format of the message.
- After completing the body of the message, the developer calls an HLO API to send the message. HLO will:
    - Build a header segment for the message based on the input parameters provided by the developer and various configuration tables.
    - Store the body of the message in the HLO MESSAGE BODY file (#777) and store the message header and administrative information in the HLO MESSAGES file (#778).
    - Place the message on an outgoing queue for an HLO client process to transmit to the remote server.

<u>Step 3:</u> The developer needs to implement the necessary event triggers in the application software. That entails inserting one or more hooks into the application that will execute the routine developed in Step 2 when the event occurs. In a typical VistA application, the hooks may be inserted directly into an application routine, option, protocol, M-style cross-reference, or other software component. How best to implement the event triggers requires careful technical analysis of the application software.

The HLO APIs support two styles of message building, which we'll refer to as "traditional style" and "new style."

Traditional style – This is the original style provided by the HL7 package prior to the development of HLO. Its characteristics are:

- Messages are built by the application outside of the HL7 package as an array of lines of text. The application typically builds each segment by concatenating the individual fields together or by setting the fields into the segment using the M $PIECE function. The completed message array is then passed to the HL7 package.
- Event protocols and subscriber protocols are created during development. These protocols define various message parameters, such as message type, event type, sending and receiving application, and receiving facility. When an application's trigger event occurs, the application passes the event protocol to the HL7 package, along with the message array described above.

New style – HLO provides applications with a new style of generating their messages. Its characteristics are:

- The application uses the HLO APIs within a framework to both build and deliver the message. The overview of the framework is:
    1. Start the process of generating a message by calling either $$NEWMSG^HLOAPI for an individual message or $$NEWBATCH^HLOAPI for a batch message.
    2. Build the message segment-by-segment and field-by-field using special HLO APIs.
    3. Once the message is complete, send it via one of the HLO SEND APIs.
- The application developer isn't required to set up protocols in advance to define various message parameters. Instead, the application can pass HLO those parameters through the formal parameter list to the HLO API.

Why support two styles of building messages?

There are many existing applications that have invested considerable effort in developing hard-coded segment builders, message builders, and protocols in the traditional style. To protect that investment, HLO supports the traditional style of message building.

On the other hand, there are distinct advantages to the new style:

- HLO relieves the developer of considerable overhead:
  - HL7 message delimiters that occur within the data are automatically replaced with their corresponding escape sequences.
  - HLO automatically keeps track of how long a segment is and prevents nodes from exceeding the maximum size allowed.
  - HLO automatically caches the message being built in memory, moving it to disk when it is complete or it grows too large for the symbol table.
- The message building APIs are also designed to make message building self-documenting and resistant to the types of errors that tend to occur in hard-coding complex segments via piecing together individual fields.

- The setup without the use of protocols can be simpler and more flexible.

So while HLO supports both styles, the downside is that HLO has more APIs than otherwise.

2.2 Create an Entry in the HLO Application Registry File for the Sending Application

The application must specify a name for the sending application when generating messages to send. The application name is required to be unique within the enterprise, so it should be namespaced by the sending package. The sending application appears in the message header segments MSH-4 or BHS-4.

HLO requires that a sending application have an entry in the HLO APPLICATION REGISTRY file (#779.2), but the information required is minimal.

Using the ADD/EDIT SENDING APPLICATION option [HLO EDIT SENDING APPLICATION] on the APPLICATION REGISTRY MENU option [HLO APPLICATION REGISTRY MENU], which in turn is under the HL7 (Optimized) MAIN MENU option [HLO MAIN MENU], create an entry for the sending application. Only the name and package are required for the sending application.

Example: Adding a sending application to the HLO Application Registry. In this example the sending application will not use sequence queues.

```
Select HLO APPLICATION REGISTRY APPLICATION NAME: HLO DEMO SENDING APPLICATION
 Are you adding 'HLO DEMO SENDING APPLICATION' as
  a new HLO APPLICATION REGISTRY (the 69TH)? No// Y (Yes)
APPLICATION NAME: HLO DEMO SENDING APPLICATION Replace
Package File Link: HLO HL7 OPTIMIZED (HLO)   HLO
Do you want to edit the setup for sequence queues? NO//
```

Example: Adding a sending application to the HLO Application Registry. In this example, the sending application will use sequence queues (see section 2.7).

```
Select HLO APPLICATION REGISTRY APPLICATION NAME: HLO DEMO SENDING APPLICATION
 Are you adding 'HLO DEMO SENDING APPLICATION' as
  a new HLO APPLICATION REGISTRY (the 69TH)? No// Y (Yes)
APPLICATION NAME: HLO DEMO SENDING APPLICATION Replace
Package File Link: HLO HL7 OPTIMIZED (HLO)   HLO
```

```
Do you want to edit the setup for sequence queues? NO// YES
SEQUENCE EXCEPTION ROUTINE: HLOSEQ
SEQUENCE EXCEPTION TAG: LATE
SEQUENCING TIMEOUT: 10
```

As of patch XU*8.0*399, KIDS supports the inclusion of entries in the HLO Application Registry in software distributions by allowing the developer to include them in a build in the same way as other components such as options and protocols.

2.3 Create Entries in the HL Logical Link File for Each Destination

An outgoing message must be associated with an entry in the HL LOGICAL LINK file (#870) in order to be transmitted. The HL Logical Link provides the IP internet address, port number, domain, and if applicable, the VHA station number.

An HLO server can coexist with other servers at the same IP address, but it must have its own port reserved exclusively for HLO. The VHA DBA has assigned standard ports for HLO as follows:

Production Systems:	Port 5001
Development and Test Systems:	Port 5026

The field that identifies the HLO port is the TCP/IP PORT (OPTIMIZED) (#400.08). If not valued, HLO will default to the standard port shown above. HLO determines whether it's a production system via the HLO SYSTEM PARAMETERS file (#779.1).

The majority of fields defined for the HL LOGICAL LINK file (#870) are defined for the use of the HL7 1.6 and are not used by HLO. The following fields *are* used by HLO:

Fields Used by HLO in the HL Logical Link file.	
NODE (#.01)	The name of the HL LOGICAL LINK entry, file #7870.
INSTITUTION (#.02)	If there is an entry in the INSTITUTION file (#4) for the remote facility then set this field. It will be used to obtain the station number.
LLP TYPE (#2)	Set this to "TCP".
TCP/IP SERVICE TYPE (#400.03)	Set this to "CLIENT".
MAILMAN DOMAIN (#.03)	Not used if the DNS DOMAIN field (below) is defined. Otherwise, set this to an entry in the MailMan's DOMAIN file (#4.2).
DNS DOMAIN (#.08)	Set this to the TCP/IP domain name of the remote server.
TCP/IP ADDRESS (#400.01)	Set this to the TCP/IP address of the remote server.
TCP/IP PORT (OPTIMIZED) (#400.08)	Set to the port number used by the remote server.

Every system on the network should have an official domain name assigned. Using the domain name in conjunction with the Dynamic Name Service (DNS) protocol, HLO can find the TCP/IP address of the remote server – which is important in case the TCP/IP address changes. It allows HLO to automatically recover without disruption. When VHA's DNS service was created it was initially populated with domains taken from the MailMan's DOMAIN file (#4.2). HLO also uses domain names in the HL7 message headers to identify the sending and receiving facilities.

Use the Link Edit [HL EDIT LOGICAL LINKS] option to create or edit an HL Logical Link entry, setting only the fields listed above.

WARNING: If modifying an existing HL Logical Link entry for use with HLO, do NOT modify or delete any field not specific to HLO. Though HLO does not use those other fields, HL7 1.6 does.

2.4 The MSH, BHS, and BTS Segments

HLO automatically builds the MSH and BHS message header segments and the BTS batch trailer segment. The following sections provide a description of those segments.

2.4.1 HLO's MSH Segment

Sequence	Field Name	Assigned Value
1	Field Separator	Required. Defaults to "|", but another value may be provided by the application.
2	Encoding Characters	Required. Defaults to "^~\&", but another value may be provided by the application.
3	Sending Application	Required. Provided by the application.
4	Sending Facility	Component 1: the station number Component 2: the domain name Component 3: "DNS" These values are determined from the HLO SYSTEM PARAMETERS file (#779.1).
5	Receiving Application	Required. Provided by the application.
6	Receiving Facility	Component 1: the station number Component 2: the domain name Component 3: "DNS" These values are determined from the HL LOGICAL LINK file (#870).
7	Date/Time Of Message	Required. Set by HLO at the point the message is generated by the application.
8	Security	Optional. Provided by the application.

9	Message Type	Required. A three-letter code provided by the application.
10	Message Control ID	Required. The value is based on a counter and is prefixed with the station number to insure its uniqueness within VHA.
11	Processing ID	Required. The value is determined by the HLO SYSTEM PARAMETERS file (#779.1)
12	Version ID	Required. Defaults to 2.4, but the application may provide another value.
13	Sequence Number	Not used.
14	Continuation Pointer	Optional. Provided by the application.
15	Accept Acknowledgment Type	Required. “NE” or “AL” as specified by the application.
16	Application Acknowledgment Type	Required. “NE” or “AL” as specified by the application.
17	Country Code	Optional. Three-character code provided by the application.
18	Character Set	Not used.
19	Principal Language Of Message	Not used.
20	Alternate Character Set Handling Scheme	Not used.
21	Message Profile Identifier	Not used.

2.4.2 HLO’s BHS Segment

Sequence	**Field Name**	**Assigned Value**
1	Batch Field Separator	Required. Defaults to “|”, but another value may be provided by the application.
2	Batch Encoding Characters	Required. Defaults to “^~\&”, but another value may be provided by the application.
3	Batch Sending Application	Required. Provided by the application.
4	Batch Sending Facility	Component 1: the station number Component 2: the domain name Component 3: “DNS” The values are determined from the HLO SYSTEM PARAMETERS file (#779.1).

5	Batch Receiving Application	Required. Provided by the application.
6	Batch Receiving Facility	Component 1: the station number Component 2: the domain name Component 3: “DNS” The values are determined from the HL LOGICAL LINK file (#870).
7	Batch Creation Date/Time	Required. Set by HLO at the point the message is generated by the application.
8	Batch Security	Optional. Provided by the application.
9	Batch Name/ID/Type	This field is used by HLO for a non-standard purpose. It is used to store three items of information, the Processing ID, determined from the HLO SYSTEM PARAMETERS file (#779.1), the Accept Acknowledgment Type , provided by the application, and the Application Acknowledgment Type, provided by the application. The format of the field is: “PROCESSING ID=”<Processing ID code is “P” or “D”> “ACCEPT ACK TYPE=”<Accept Acknowledgment Type is “NE” or “AL”> “APP ACK TYPE=”<Application Acknowledgment Type is “NE” or “AL”>
10	Batch Comment	Not used.
11	Batch Control ID	Required. The value is based on a counter and is prefixed with the station number to insure its uniqueness within VHA.
12	Reference Batch Control ID	This is used only for a batch that contains application acknowledgments to another batch. It is set to the Batch Control ID of the original batch.

2.4.3 HLO’s BTS Segment

Sequence	**Field Name**	**Assigned Value**
1	Batch Message Count	Required. The value is set to a count of the messages within the batch.
2	Batch Comment	Not used.
3	Batch Totals	Not Used.

2.5 List of Message Building APIs

New Style Message Building Group 1	
Building a Segment Group 1.1	
Inserting Data into a Segment Group 1.1.1 • Not all data types in the HL7 standard are supported by a specific API. • These APIs automatically replaces message delimiters with escape sequences.	
SET^HLOAPI	Sets a single atomic value of unspecified data type into the segment.
SETAD^HLOAI4	Sets an address into a segment
SETCE^HLOAPI4	Sets a coded element into a segment.
SETCNE^HLOPAI4	Sets a coded value with no exceptions into a segment.
SETCWE^HLOAPI4	Sets a coded value with exceptions into a segment.
SETDT^HLOAPI4	Sets a date into the segment.
SETHD^HLOAPI4	Sets a hierarchic designator into a segment.
SETTS^HLOAPI4	Sets a timestamp into the segment.
SETXPN^HLOAPI4	Sets an extended person name into a segment.
Inserting the Completed Segment Into the Message Group 1.1.2	
ADDSEG^HLOAPI	Adds the completed segment to the end of the message.
Building an Individual Message Group 1.2	
NEWMSG^HLOAPI	Begins a new message.
Building a Batch Message Group 1.3	
NEWBATCH^HLOAPI	Begins a new batch of messages.
ADDMSG^HLOAPI	Adds another message into the batch.
Sending a Batch Message or Individual Message whose Construction is Complete Group 1.4 • Stores the body of the message in the HLO MESSAGE BODY (#777) • For each recipient, creates an entry in the HLO MESSAGES (#778) • The message is placed on an outgoing queue for transmission for each recipient.	
SENDONE^HLOAPI1	Sends a message to a single destination.
SENDMANY^HLOAPI1	Sends a message to a list of destinations, passed in as a parameter.
SENDSUB^HLOAPI1	Sends a message to a list of destinations. The list is based on an entry in the HLO Subscription Registry.
Traditional Style Message Building Group 2	
EN^HLOCNRT	Allows the use of traditional-style hardcoded message builders and protocol setup as in HL7 1.6, but the messages are sent via HLO. It is similar to HL7 1.6's EN^HLMA. However, it does not support the sending of batch

				messages, the use of the ROUTING LOGIC field of a protocol, and other features of EN^HLMA.
Hybrid Style Message Building Group 3				
			MOVEMSG^HLOAPI	Moves a message array built in the traditional way into HLO. It does not require the use of protocols or other traditional setup.
			MOVESEG^HLOAPI	Moves a single segment built in the traditional way into HLO. This preserves the investment in the library of message builders that were already developed prior to HLO.

2.6 The HLO Framework for Message Building

The following set of patterns defines the framework under which applications may use the HLO APIs to generate their messages.

Message building within HLO always requires the following steps, so they won't be included when discussing the individual patterns:

1. Setting up entries in the HL LOGICAL LINK file (#870) for the network destinations for the messages
2. Setting up entries in the HLO APPLICATION REGISTRY file (#779.2) for the sending application
3. Using the M NEW command to set up the required variables used for calls to the various APIs

Also, the technical details for using each API are not presented here. For that, please refer to the API Reference Guide.

2.6.1 New-Style Message Building

2.6.1.1 Pattern 1 – Building a Segment

The APIs in group 1.1.1 are used iteratively to build a segment, field by field.

- The fields may be set in any order, though left to right is recommended.
- The particular SET-type API used depends on the data type of the field, component, or subcomponent. HLO does not provide specialized APIs for all of the HL7 standard's complex data types, but the generic SET^HLOAPI can always be used to set a single atomic value into a segment at the field, repetition, component, or subcomponent level.

Pattern:

- Start building the segment by setting the three-character segment type into your workspace (SEG):
  - DO SET^HLOAPI(.SEG,<3 character segment type>)

    or equivalently:
  - DO SET^HLOAPI(.SEG,<3 character segment type>,0)

- For each field in the segment that is to contain a data value:
    - Call the APIs from group 1.1.1 to set the data into the field. If the field is repeating, or has multiple components or subcomponents, you may need to make multiple calls.

Note: In the following pages, the numbered examples refer to the code examples in the section on Code Examples. The un-numbered examples are self contained.

Example 1: Building a PID segment using the HLO APIs. See PID^HLODEM1 in the section on Code Examples. Segments built this way are added to the message by calling ADDSEG^HLOAPI.

Example 2: For the sake of comparison, this example builds the same PID segment *without* using the HLO APIs. See PID^HLODEM2 in the section on Code Examples. Segments built this way are added to the message by calling MOVESEG^HLOAPI.

Example 3: Builds a NK1 segment using the HLO APIs. See NK1^HLODEM1 in the section on Code Examples.

Example 4: This example builds the same NK1 segment *without* using the HLO APIs.

2.6.1.2 Pattern 2 – Building an Individual Message

This builds a complete individual message, as opposed to a batch message, and places it on an outgoing queue for transmission. The message header segment MSH is automatically added to the message.

<u>Pattern:</u>

- Call NEWMSG^HLOAPI to start building the message.
- For each segment to appear in the message, other than the MSH segment, in the correct order defined for its message structure based on the message type and event:
    - Build the segment using pattern 1.
    - Call ADDSEG^HLOAPI to add the completed segment to the message.
- Call one of the APIs in group 1.4 to complete the message and queue it for transmission.

Example 5: Builds an ADT~A08 message and queues it for transmission. See A08^HLODEM1 in the section on Code Examples.

2.6.1.3 Pattern 3 – Building a Batch Message

This builds batch message, and places it on an outgoing queue for transmission. The batch header segment BHS and individual message header segments MSH are automatically added to the message.
Pattern:

- Call NEWBATCH^HLOAPI to start the batch.
- For each message to be placed in the batch:
    - Call ADDMSG^HLOAPI to start the message.
    - For each segment to appear in the message, other than the MSH segment, in the correct order defined for its message structure based on the message type and event:
        - ❖ Build the segment using pattern 1.
        - ❖ Call ADDSEG^HLOAPI to add the completed segment to the message.
- Call one of the APIs in group 1.4 to send the message.

Example 6: Builds a batch of ADT~A08 message and queues it for transmission. See BATCHA08^HLODEM1 in the section on Code Examples.

2.6.2 Traditional Style Message Building

2.6.2.1 Pattern 4 Building an Individual Message in the Traditional Style

So that existing applications might easily convert from HL7 1.6 to HLO, HLO supports traditional style hard-coded message builders and the use of protocols as found in HL7 1.6. HLO does NOT support sending batch messages built with HL7 1.6 tools.

<u>Pattern:</u>

- Set up entries in the HL7 APPLICATION PARAMETER file (#771) for the sending and receiving applications.
- Set up an event protocol and its subscriber protocols.
- Create the body of the message in either the HLA("HLS") local array or the ^TMP("HLS",$J) global array in the traditional way.
- Call EN^HLOCNRT.

Example 7: Builds an ADT~A08 using a hardcoded message builder and a protocol setup. The resultant message is identical to that of example 4. See A08^HLODEM2 in the section on Code Examples.

2.6.3 Hybrid Style Message Building

There are several combinations in which elements of the traditional style message building may be mixed with the new style message building.

2.6.3.1 Pattern 5 – Mixing Traditional Style and New Style Segment Builders

A message can be built using a mix of the new segment building APIs and the traditional style segment builders.
Pattern:

- Follows new style Patterns 2 or 3, except for each segment:
  - If the segment was built via a traditional segment builder as an array of lines, add it to the message by calling MOVESEG^HLOAPI.
  - If the segment was built using the HLO segment building APIs, add it to the message by calling ADDSET^HLOAPI.

Example 8: Builds an ADT~A08 using a the HLO segment building APIs for the PID segment and a traditional hardcoded segment builder for the NK1 segment. The resultant message is identical to that of example 4. See A08^HLODEM3 in the appendix of code examples.

2.6.3.2 Pattern 6 – Traditional Message Building without Protocols

Messages built using a traditional hardcoded message builder can be sent via HLO without using protocols.

Pattern:

- Create an entry in the HLO Application Registry for the sending application.
- Create the individual message in either the HLA ("HLS") local array or the ^TMP("HLS",$J) global array in the traditional way.
- Call NEWMSG^HLOAPI to start building the message.
- Call MOVEMSG^HLOAPI to move the body of the message, built in the tradition style as an array of segments, into the message started by the call to NEWMSG^HLOAPI.
- Call one of the APIs in group 1.4 to complete the message and queue it for transmission.

Example 9: Builds an ADT~A08 using with a hardcoded message builder but sends it without the protocol setup. See A08^HLODEM3 in the appendix of code examples.

2.7 Application Callbacks

A callback is an application routine that is executed upon the occurrence of some future event. With HLO, when an application sends a message, it may set three different callbacks as parameters.

2.7.1 Callback for Application Acknowledgments

The application may set a callback that will be executed when the application acknowledgment message is returned. To do so, set this parameter before calling one of the send APIs listed in group 1.4 of section 2.5:

S PARMS("APP ACK RESPONSE") = <tag>^<routine>

At the point the routine is executed by HLO, the only variable defined is HLMSGIEN equal to the IEN of the application acknowledgment. As always, the application should start by calling $$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.HDR). The array MSG is returned with information about the application acknowledgment, but to refer back to the original message the variable MSG("ACK TO") is returned with the Message Control ID of the original message and MSG("ACK TO IEN") is returned with the IEN of the original message in the HLO MESSAGES file (#778).

2.7.2 Callback for Commit Acknowledgments

The application may set a callback that will be executed when the commit acknowledgment message is returned. To do so, set this parameter before calling one of the send APIs listed in group 1.4 of section 2.5:

S PARMS("ACCEPT ACK RESPONSE") = <tag>^<routine>

At the point the routine is executed by HLO, the only variable defined is HLMSGIEN set equal to the IEN of the *original* message. That is because the commit acknowledgment is not stored separately. Instead, when the commit acknowledgment is returned, HLO does the following:

- If the commit acknowledgment returns an error or reject, the status of the original message is set to error.
- The MSA segment of the commit acknowledgment is stored with the original message.

As with the callback for the application acknowledgment, the application callback should start off by calling $$STARTMSG^HLOPRS(.MSG,.HLMSGIEN,.HDR), but in this case the MSG array is returned with information pertaining to the *original* message. Two items of information that are of particular importance when processing a commit acknowledgment in a callback:

- MSG(“STATUS”) will be “ER” if the commit acknowledgment indicated an error or reject.
- MSG("STATUS","ACCEPT ACK ID") is the message id of the commit acknowledgment that was returned.
- MSG(“MSG("STATUS","ACCEPT ACK MSA") will be the MSA segment of the commit acknowledgment that was returned.

2.7.3 Callbacks for Messages that Fail to Transmit

HLO will repeatedly attempt to transmit a message until its either successful or a negative acknowledgment is returned. But eventually, after a very long period determined by the site, HLO will cease its attempts to transmit a message and set its status to “error.” The sending application can set a callback to be executed in this eventuality by setting this parameter:

PARMS(“FAILURE RESPONSE”)=<tag>^<routine>

At the point, the callback is executed; the only variable defined is HLMSGIEN equal to the IEN in the HLO MESSAGES file (#778) of the message.

2.8 Sequence Queues

Sequence queues are a mechanism for ensuring that messages are received by the remote application in the same order that the sending application generates them. When sending messages between two VistA systems with no middleware involved, sequence queues are NOT necessary. But when the sending application uses the services of the VIE or other middleware, or when sending messages to a non-VistA system, sequence queues *may* be necessary to insure the sequence of message delivery.

2.8.1 The Algorithm

Sequence queues work in conjunction with the use of application acknowledgments to insure the correct sequence of message delivery. The algorithm has two parts, with roles for both the client and the server.

2.8.1.1 Client Role:

- At the point when a sending application generates a message, it requests:
  a. That an application acknowledgement be returned.

  b. That the message be placed on its private sequence queue.

- If the sequence queue is NOT currently pending the return of an application acknowledgment for a preceding message, then HLO places the new message on the outgoing queue and marks the sequence queue as pending the return of an application acknowledgment for this new message.

- However, if the sequence queue IS currently pending the return of an application acknowledgment for the preceding message, then HLO does NOT place the new message on the outgoing queue. Instead, the new message is placed on the sequence queue, pending the application acknowledgment for the preceding message.

2.8.1.2 Server Role:

- The server receives the return of an application acknowledgment.
- It looks up the original message to determine whether or not the sending application had used a sequence queue.
- If so, it goes to that sequence queue and moves the next message on it into the outgoing queue for transmission, and marks the sequence queue as pending an application acknowledgment for that message.

2.8.2 Using Sequence Queues

To use a sequence queue, an application developer needs to take these additional steps in building their messaging application:

1. The sending application must request both an accept acknowledgment and an application acknowledgement.
2. The sending application must specify the name of a sequence queue on which to place the message. The queue is created dynamically as needed, and disappears once it is empty. The name of the queue can be up to 30 characters and **must be namespaced** by the application. It is passed as an input parameter to any of the HLO message sending APIs by specifying PARMS("SEQUENCE QUEUE"). The input parameter must be passed-by-reference.
3. The application must provide an exception handler for HLO to invoke if a sequence queue waits too long for an application acknowledgment to be returned. Typically, the exception handler will serve to notify the operations staff to investigate why the remote application has not returned the application acknowledgement. The queue may be manually advanced if it is determined to be the appropriate action.
4. There are new parameters that must be entered for the sending application in the HLO Application Register file (#779.2).
    a. SEQUENCE TIMEOUT field (#.12) - The number of minutes to wait for an application acknowledgment before calling the application's sequence exception routine.

    b. SEQUENCE EXCEPTION ROUTINE field (#.10) and SEQUENCE EXCEPTION TAG field (#.11) - Together these two fields specify an M <tag>^<routine> to call when a the timeout period expires while waiting for an application acknowledgment.

Example 10: This is identical to example 5, A08^HLODEM1, except that the message is placed on a sequence queue. See A08^HLODEM4 in the section on Code Examples.

2.9 HLO Subscription Registry

The HLO Subscription Registry (file #779.4) is used to store lists of subscribers to an application’s messages. Conceptually, it’s the same as the subscriber list stored in the PROTOCOL file (#101) used by HL7 1.6, or distribution lists used by email. An application may address a message to a subscription list, and HLO will send the message to each subscriber on that list.

HLO has two interfaces to the subscription registry. It has a user interface which is used to create a subscription list at design time. Subscription lists created that way are distributed along with the other system components of the messaging application via KIDS.

The other interface is via a set of APIs used to create subscriptions on-the-fly on the system where they will be used. For example, if a new patient is registered at a site, an application may create a subscription specifically related to that patient to keep informed of that patient's activities via HL7 messages.

2.9.1 The User Interface for Creating Subscription Lists

The ADD/EDIT SUBSCRIPTIONS option is used to create a subscription list and to add subscribers to it. The option is under the Developer Menu, which in turn is under the HLO MAIN MENU.

These fields should be entered to create an entry in the HLO Subscription Registry (file #779.4).

#	Field Name	Description
.01	NAME	Then name of the entry. It must be unique, so it should be namespaced.
.02	OWNER APPLICATION	Could be the package, the sending application, or other meaningful designation for the creator.
.03	DESCRIPTION	Optional. A brief one-line description.

For each subscriber to be added, these fields should be entered to the RECIPIENTS sub-file (#794.41).

#	Field Name	Description
.01	RECEIVING APPLICATION	The name of the receiving application, exactly as it should appear in the message header.
.02	MIDDLEWARE LOGICAL LINK	Enter this field ONLY IF the message should be transmitted through middleware, such as an interface engine. It is the HL Logical Link file (#870) that has the IP address and port # of the middleware.
.021	FACILITY LOGICAL LINK	This field is ALWAYS needed. It is the HL Logical Link of the facility that the message is being sent to. It is used to obtain values for the Receiving Facility field of the message header. If a MIDDLEWARE LOGICAL INK is not specified, it is also used to obtain the IP address and port # for where to transmit the message.

Example: Adding a new entry to the HLO Subscription Registry using the ADD/EDIT SUBSCRIPTIONS option. One subscriber is added to the new subscriber list.

```
Select HLO DEVELOPER MENU Option: ADD/EDIT SUBSCRIPTIONS

Select HLO SUBSCRIPTION REGISTRY NAME: HLO DEMONSTRATION APPLICATION
 Are you adding 'HLO DEMONSTRATION APPLICATION' as a new HLO SUBSCRIPTION REGISTRY (the
21ST)? No// Y (Yes)
```

```
NAME: HLO DEMONSTRATION APPLICATION Replace
OWNER APPLICATION: HLO
DESCRIPTION: SUBSCRIPTION LIST FOR HLO’S DEMONSTRATION APPLICATION
Select RECEIVING APPLICATION: HLO DEMO RECEIVING APPLICATION
 Are you adding ’HLO DEMO RECEIVING APPLICATION' as
  a new RECEIVING APPLICATION (the 1ST for this HLO SUBSCRIPTION REGISTRY)? No// YES
 MIDDLEWARE LOGICAL LINK:
 RECEIVING FACILITY LOGICAL LINK: VAALB
 DATE/TIME TERMINATED:
Select RECEIVING APPLICATION:
```

In the example above, since the messages won’t be routed through middleware such as an interface engine, the MIDDLEWARE LOGICAL LINK was not entered.

Entering a DATE/TIME TERMINATED indicates that the subscription is terminated and would cause messages to that particular subscriber to stop.

2.9.2 The Programming Interface for Creating Subscription Lists

An application may create a subscription list programmatically. For example, it could be used to send a patient’s health data to a special registry if an indicator is present. The following framework explains how to use the subscription API’s to do that.

<u>Step 1:</u> Create an entry in the HLO SUBSCRIPTION REGISTRY file (#779.4).

This is accomplished by calling $$CREATE^HLOASUB(OWNER,DESCRIPTION,.ERROR). The function returns the IEN of the newly created entry. The NAME field (#.01) is automatically set to the IEN.

Example: Creating a new subscription list.

```
 S IEN=$$CREATE^HLOASUB(“HLO DEMO APPLICATION”,”PATIENTS IN SPECIAL REGISTRY”)
```

<u>Step 2:</u> (Optional) Add lookup values for the new entry.

In step 1, an IEN is returned for the new subscription list. An application may store that IEN, in which case lookup values are not necessary, though still useful. However, if the IEN is *not* stored by the application, then it *must* add lookup values to uniquely identify the subscription list, because otherwise it would be unable to find the subscription.

Lookup values are added by calling $$INDEX^HLOASUB1(IEN,.LOOKUP)

If used, the application may add up to five lookup values. They will be used to create an index that will enable the application to find the subscription. If the owner name is not sufficiently unique, then at least one of the lookup values must be namespaced to insure its uniqueness. If the subscription is patient specific, at least one of the lookup values must contain a unique patient identifier, such as the DFN or ICN.

Example: Adding lookup values for the subscription list, where IEN=the IEN of the entry just created, and DFN is the patient DFN.

```
N LOOKUP
S LOOKUP(1)=”HLO PATIENT REGISTRY”
S LOOKUP(2)=DFN
I $$INDEX^HLOASUB(IEN,.LOOKUP) ;then success
```

After doing this, the application can find the subscription list by using $$FIND^HLOSUB1(OWNER,.LOOKUP).

Example: Finding the subscription using the lookup values.

```
N LOOKUP,IEN
S LOOKUP(1)=”HLO PATIENT REGISTRY”
S LOOKUP(2)=DFN
S IEN=$$FIND^HLOASUB1(“HLO DEMO APPLICATION”,.LOOKUP)
```

<u>Step 3:</u> Adding subscribers to the subscriber list.

For each destination where these messages should be sent, call $$ADD^HLOASUB(IEN,.WHO,.ERROR) to add that destination as a subscriber. The WHO parameter is an array that is used to identify the name of the receiving application and the receiving facility, and a link over which to send the message.

Example: Adding a subscriber to the list, where IEN is the IEN of the list.

```
N WHO
S WHO(“RECEIVING APPLICATION”)=”HLO DEMO RECEIVING APPLICATION”
S WHO(“STATION NUMBER”)=500
S WHO(“MIDDLEWARE LINK NAME”)=”VAVIE”
I $$ADD^HLOASUB(IEN,.WHO,.ERROR) ;then the subscriber was successfully added, else ERROR would contain
an error message.
```

The identification of a link for middleware in the WHO parameter array is optional and only necessary if the messages will go through middleware such as an interface engine. Also, there are several choices for how to identify the receiving facility, either by station number, IEN of an entry in the INSTITUTION file (#4), or an entry in the HL LOGICAL LINK file (#870).

2.10 Turning Messaging On/Off

An application may need to build in the capability of turning its message generation on or off. HLO provides some support for that via the HLO Application Registry and the INACTIVATE SENDING APPLICATION option, which is on the APPLICATION REGISTRY MENU [HLO APPLICATION REGISTRY MENU] under the DEVELOPER MENU [HLO DEVELOPER MENU]. Using the INACTIVATE SENDING APPLICATION option [HLO TERMINATE OUTGOING MESSAGE] a user may turn on or off a sending application’s messages.

Example: Using the INACTIVATE SENDING APPLICATION option to set the ADT~A08 messages inactive flag.

```
Select APPLICATION REGISTRY MENU Option: INACTIVATE SENDING APPLICATION

Select HLO APPLICATION REGISTRY APPLICATION NAME: HLO DEMO SENDING APPLICATION
```

```
APPLICATION NAME: HLO DEMO SENDING APPLICATION Replace
Select HL7 MESSAGE TYPE: ADT
 Are you adding 'ADT' as a new HL7 MESSAGE TYPE (the 1ST for this HLO APPLICATION REGISTRY)? No// Y (Yes)
  HL7 MESSAGE TYPE HL7 EVENT: A08
  HL7 MESSAGE TYPE HL7 VERSION:
 HL7 EVENT: A01//
 HL7 VERSION:
 INACTIVE: INACTIVE
```

Note: The application must be designed to honor the inactive flag - otherwise it will have no effect!

To enable this ability, the application must check the flag within its event trigger code. If the flag is set to INACTIVE, the application should quit without generating the message. The flag is checked by calling $$ACTIVE^HLOAPP(APP,MSGTYPE,EVENT,VERSION)

Example: Designing the sending application to honor the INACTIVE flag.
This example refers to A08^HLODEM1 in the section on Code Examples. To build in an ON/OFF switch for the ADT~A08 message, the application could insert the following line of code at the A08^HLODEM1.

```
A08     ;
        Q:'$$ACTIVE^HLOAPP("HLO DEMO SENDING APPLICATION","ADT","A01")
```

3.0 Receiving Messages

Applications receive their messages by following these steps:

1. The receiving application must be registered in the HLO Application Registry (file # 779.2).
2. The HLO server process follows the algorithm described in section 1.4.2 Server Behavior. It reads the message and parses the header.
3. Using the name of the receiving application, type of message (BHS or MSH), message type, event type, and version, it does a lookup on the HLO Application Registry (file 779.2) to determine the application routine that should be executed to process the message.
4. If the lookup fails, and a commit acknowledgment was requested, the HLO server will return an error to the sending system. The message status is set to error and appears in HLO's log of message errors.
5. If the lookup succeeds, the server places the message on an incoming queue, with information about the application routine to execute.
6. Another HLO process operates continuously in the background, looking for new messages on the incoming queue. For each new message, it executes the application routine that was specified in the HLO Application Registry for the receiving application.
7. The application routine is responsible for parsing the message and taking appropriate actions. HLO provides a set of APIs for parsing the message and for returning application acknowledgments.

3.1 Setting up the Receiving Application in the HLO Application Registry

In order for an application to receive messages, it must be entered in the HLO APPLICATION REGISTRY file (#779.2) at the site. The primary purpose is to designate which application routine should be executed to process which incoming message. The application may designate a default message handler, a message handler for batch messages, and specific message handlers for specific messages. The application may also designate private queues to place the messages, the alternative is that the messages are by default placed on an incoming queue named ’DEFAULT’.

Using the ADD/EDIT RECEIVING APPLICATION option [HLO EDIT RECEIVING APPLICATION] on the APPLICATION REGISTRY MENU [HLO APPLICATION REGISTRY MENU], which in turn is under the HL7 (Optimized) MAIN MENU option [HLO MAIN MENU], create an entry for the receiving application.

Applications in the HLO Application Registry (file 779.4) can be included in a KIDS build as other components such as protocols, and included in software distributions.

Note: The name of the receiving application must be unique within the enterprise. Wherever possible, the name should be namespaced.

Example: Adding a receiving application to the HLO Application Registry.

```
Select APPLICATION REGISTRY MENU Option: ADD/EDIT RECEIVING APPLICATION

Select HLO APPLICATION REGISTRY APPLICATION NAME: HLO DEMO RECEIVING APPLICATION
 Are you adding ’HLO DEMO RECEIVING APPLICATIONS' as a new HLO APPLICATION REGISTRY (the
70TH)? No// Y (Yes)
APPLICATION NAME: HLO DEMO RECEIVING APPLICATIONS
     Replace
Package File Link: HLO HL7 OPTIMIZED (HLO)   HLO
Do you want to edit the setup for receiving batch messages? NO// YES
BATCH ACTION ROUTINE: HLODEM7
BATCH ACTION TAG: BATCH
If a private queue is used for batches its name must be namespaced!
BATCH PRIVATE IN-QUEUE: HLO DEM BATCH
Select HL7 MESSAGE TYPE: ADT
 Are you adding ’ADT' as a new HL7 MESSAGE TYPE (the 1ST for this HLO APPLICATION REGISTRY)? No//
Y (Yes)
  HL7 MESSAGE TYPE HL7 EVENT: A08
  HL7 MESSAGE TYPE HL7 VERSION:
 ACTION ROUTINE: HLODEM5
 ACTION TAG: PARSEA08
If a private queue is used the name must be namespaced!
 PRIVATE QUEUE (optional): HLO DEMO A08
Select HL7 MESSAGE TYPE:
Do you want to edit the default action for non-specified messages types? NO// YES
DEFAULT ACTION ROUTINE: HLODEM5
DEFAULT ACTION TAG: PARSEMSG
If you specify a private queue for the default action then its name must be namespaced!
DEFAULT PRIVATE QUEUE (optional):
If application acknowledgments are returned, and they must go through a specific link, enter it here. This applies if an
interface engine or other middleware is used.
RETURN LINK FOR APPLICATION ACKNOWLEDGMENTS:
```

3.2 List of Message Parsing APIs

STARTMSG^HLOPRS	Begin parsing the message. Parsing a message always starts with a call to this API. The message header fields are returned to the application.
NEXTMSG^HLOPRS	Moves the parser to the next message within the batch. Applies only to batch messages.
NEXTSEG^HLOPRS	Moves the parser to the next segment within the individual message. Applies to parsing both individual messages and batch messages. The individual values are parsed out from the segment, and escape sequences are automatically replaced with the original characters.
APIs for Parsing HL7 Data Types from a Segment These APIs are always invoked after a call to NEXTSEG^HLOPRS to obtain data from the returned segment. The application specifies the field, component, subcomponent, and repetition.	
GET^HLOPRS	Parses an individual value from a segment. Use this if HLO does not provide a special API to parse the data type. If the data type contains multiple parts, call this API multiple times, once for each part.
GETAD^HLOPRS2	Parses an address, data type AD, from a segment.
GETCE^HLOPRS2	Parses a coded element, data type CE.
GETCNE^HLOPRS2	Parses a coded value with no exceptions, data type CNE.
GETCWE^HLOPRS2	Parses a coded value with exceptions, data type CWE.
GETDT^HLOPRS2	Parses a date, data type DT.
GETHD^HLOPRS2	Parses an HL7 hierarchic designator, data type HD.
GETXPN^HLOPRS2	Parses an extended person’s name, data type XPN.
Miscellaneous APIs	
HLNEXT^HLOMSG	Returns the segment as an array of lines. Example: SEGMENT(1), SEGMENT(2), etc. This is different from NEXTSEG^HLOPRS in that HLNEXT^HLOMSG does not parse the segment into its individual pieces. If using this API, it is totally up to the application to parse the data from the segment. There are no HLO APIs to assist. Applications are responsible for replacing escape sequences with the original characters. If the segment fits within a single node, the array returned will have only one subscript=1. **Example**: SEGMENT(1)
MSGID^HLOPRS	Given the IEN of the message, in the HLO MESSAGES file (#778), this

	API returns the Control ID found in the message header.

3.3 HLO Parses the Message Header

An application always begins parsing a message by calling STARTMSG^HLOPRS. A call to $$STARTMSG^HLOPRS(.MSG,<message ien, file #778>,.HDR) returns administrative information about the message, the header segment, and the header segment's parsed fields.

The following is a partial list of the MSG array subscripts returned. See section 5.2.1 for a complete list of the subscripts returned and available to the developer. (Start Parsing a Message)

MSG("BATCH")	If the segment type is BHS, it returns 1. If it is NOT a batch message, it returns 0.
MSG("ID")	If the header type is MSH, it returns the Message Control ID. If the header type is BHS, it retur the Batch Control ID.
MSG("EVENT")	The HL7 event, only returned if the segment type is MSH.
MSG("MESSAGE TYPE")	The HL7 message type. It is returned if the segment types is MSH, but NOT if it is BHS.
MSG("HDR",1)	The unparsed message header, field sequence1 through 6.
MSG("HDR",2)	The unparsed message header, field sequence 7 to the end.

In addition, HLO will return all the fields from the header segment and return them in an array, passed-by-reference in the third parameter. The next two sections specify what subscripts are returned in the case of a BHS segment and in the case of an MSH segment.

3.3.1 Parsing the MSH Segment

For an individual message, as opposed to a batch message, a call to $$STARTMSG^HLOPRS (.MSG,<message IEN, HLO MESSAGES file (#778)>,.HDR) will return the HDR array with the following subscripts. Any escape sequences found within the returned value will be replaced with the original character.

Sequence	Subscript	Description/Comment
0	"SEGMENT TYPE"	"MSH"
1	"FIELD SEPARATOR"	The field separator
2	"COMPONENT SEPARATOR" "SUBCOMPONENT SEPARATOR" "REPETITION SEPARATOR" "ESCAPE CHARACTER"	The message delimiters
3	"SENDING APPLICATION"	The name of the sending application
4	"SENDING FACILITY",1 "SENDING FACILITY",2 "SENDING FACILITY",3	First Component Second Component Third Component
5	"RECEIVING APPLICATION"	The name of the receiving application
6	"RECEIVING FACILITY",1 "RECEIVING FACILITY",2 "RECEIVING FACILITY",3	First Component Second Component Third Component
7	"DT/TM OF MESSAGE"	Converted to FileMan format
8	"SECURITY"	The security field, which is specific to the application.

9	"MESSAGE TYPE" "EVENT" "MESSAGE STRUCTURE"	First Component Second Component Third Component
10	"MESSAGE CONTROL ID"	A unique ID for the message
11	"PROCESSING ID" "PROCESSING MODE"	First Component Second Component
12	"VERSION"	The version of the HL7 standard on which the message was based
14	"CONTINUATION POINTER"	MESSAGE CONTROL ID of the message that this one continues.
15	"ACCEPT ACK TYPE"	ACCEPT ACKNOWLEDGMENT TYPE, <AL or NE>
16	"APP ACK TYPE"	APPLICATION ACKNOWLEDGMENT TYPE = <AL or NE>
17	"COUNTRY"	COUNTRY CODE

3.3.2 Parsing the BHS Segment

A call to $$STARTMSG^HLOPRS(.MSG,<message IEN, HLO MESSAGES file (#778)>,.HDR) will return the HDR array with the following subscripts. Any escape sequences found within the returned value will be replaced with the original character.

Sequence	Subscript	Description/Comment
0	"SEGMENT TYPE"	"BHS"
1	"FIELD SEPARATOR"	The field separator
2	"COMPONENT SEPARATOR" "SUBCOMPONENT SEPARATOR" "REPETITION SEPARATOR" "ESCAPE CHARACTER"	The message delimiters
3	"SENDING APPLICATION"	The name of the sending application
4 See Note	"SENDING FACILITY",1 "SENDING FACILITY",2 "SENDING FACILITY",3	First Component Second Component Third Component
5	"RECEIVING APPLICATION"	The name of the receiving application
6 See Note	"RECEIVING FACILITY",1 HEAD "RECEIVING FACILITY",2 "RECEIVING FACILITY",3	First Component Second Component Third Component
SEQ-7	"DT/TM OF MESSAGE"	Converted to FileMan Format
SEQ-8	"SECURITY"	Specific to the application
SEQ-9	"BATCH NAME/ID/TYPE" These fields are not defined by the standard within the BHS segment, but they are needed and are encoded in BHS-9 by the VistA HLO software: "PROCESSING ID" "ACCEPT ACK TYPE" "APP ACK TYPE"	
SEQ-10	"BATCH COMMENT"	
SEQ-11	"BATCH CONTROL ID"	A unique ID for the message

SEQ-12	“REFERENCE BATCH CONTROL ID”	If this batch message contains application acknowledgments from another batch, this will be the BATCH CONROL ID of the original batch.

Note: BHS-4 & BHS-6 The HL7 Version 2.4 Standards Manual does not document this, but components 2 & 3 were added in an erratum to be compatible with the MSH segment. It is documented in version 2.5 of the standard.

3.4 Stepping Through an Individual Message

Message parsing always starts with these steps:

- When a message is received, HLO executes the application’s message handler routine in the background.
- At that point, the ONLY variables defined are:
  - o HLMSGIEN set to the IEN of the message in the HLO MESSAGES file, #778.
  - o The variables are defined by calling DUZ^XUP with DUZ set to .5, which is a proxy for the Postmaster.
- The application must call $$STARTMSG^HLOPRS to start parsing the message. The API will return information about the message and the fields parsed from the header. (see section 3.3)

At this point, the application steps through the message segment by segment, parsing each for its data. But there are two different methods for parsing the individual segment.

3.4.1 Method 1 – Using $$NEXTSEG^HLOPRS

This is the preferred method for parsing segments within HLO because it automatically parses the segment into all its parts and automatically replaces escape sequences for delimiters with the original characters. Another advantage is that using this method results in routines that are self-documenting and less prone to the type of errors that occur with hard-coded parsing.

Calling $$NEXTSEG^HLOPRS accomplishes the following:

- The parser is moved forward one segment and the segment is returned. The function returns 1 if a segment is returned, and 0 if there are no more segments in the message. If called within a batch, 0 is returned if the end of the individual message is reached, even if there are more messages within the batch.
- The segment is totally parsed into its individual parts.
- If the segment contained any escape sequences for message delimiters, they are automatically replaced by the original characters.

After calling $$NEXTSEG^HLOPRS, the application then calls the data type parsing APIs (see section 3.2) to retrieve the field values.

3.4.2 Method 2 – Using $$HLNEXT^HLOMSG

This method is modeled after HL7 1.6 message parsing and is provided so that older applications may use their existing HL7 1.6 message parsers for messaging with HLO. Its use is discouraged for new applications.

Calling $$HLNEXT ^HLOMSG accomplishes the following:

- The parser is moved forward one segment and the segment is returned. The function returns 1 if a segment is returned, and 0 if there are no more messages in the segment. If called within a batch, 0 is returned if the end of the individual message is reached, even if there are more messages within the batch.
- The segment is NOT parsed. The application is responsible for parsing out the data. If the data contains escape sequences, the application is responsible for replacing them with the original characters.

There are differences between using $$HLNEXT^HLOMSG and parsing in HL7 1.6:

- In HL7 1.6, the variable HLNEXT is executed with the M EXECUTE command to move the parser to the next segment, rather than calling it as a function: X HLNEXT.
- If parsing a batch message, $$HLNEXT will NOT move the parser to the next message within the batch when the current message's end is reached.

3.4.3 Examples of Message Parsing

Example: A simple message parser.

```
N MSG,HDR,SEG
I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.HDR) {report error} QUIT
F Q:'$$NEXTSEG^HLOPRS(.MSG,.SEG) D
.I SEG("SEGMENT TYPE")={3 character segment type} D
..{use HLO's data type parsing routines to retrieve the data from the segment}
```

In this example:

- HLMSGIEN is the internal entry number of the message record in the HLO MESSAGES file (#778). If HLO is executing the application's routine in the background, then HLO will set that variable for the application's use.
- MSG stores information about the message, including the parser's current position in the message.
- HDR will contain the fields that were parsed from the message header.
- SEG will contain the fields that were parsed from the current segment.
- Parsing always starts off with a call to $$STARTMSG^HLOPRS. It returns both the MSG array and the HDR array. If the function returns 0, then something is wrong – the message record has been deleted or is corrupted!
- After calling $$STARTMSG^HLOPRS, the HLO parser is pointing to the message header. Calling $$NEXTSEG^HLOPRS iteratively will step the parser through the message's other segments.
- After each call to $$NEXTSEG^HLOPRS, the HLO parser will return with the segment, parsed into its fields, components, and subcomponents, with any escape sequences replaced with the original characters.
- At that point, the application uses the HLO data type parsing APIs to extract the needed data.
- If $$NEXTSEG^HLOPRS returns 0, it means that there are no more segments left in the message.

Example 11: In this example, the set up for the receiving application in the HLO APPPLICATION REGISRY file (#779.2) does not include a specific handler for the ADT~A08 message, but does specify a default message handler. When an ADT~A08 message is received, HLO will execute the default handler and it is up to the application to determine what specific action to take based on the message type and event.

Note that the HLO APPLICATION REGISTRY specifies that HLO should place this application's messages on a private queue named 'HLO DEMO CLIENT'. That is an optional feature. If a private queue was not specified, then the application's messages would be placed on a generic queue named "DEFAULT."

See PARSEMSG^HLODEM5 in the section on Code Examples.

Example 12: In this example, the set up for the receiving application in the HLO APPPLICATION REGISRY file (#779.2) does specify a message handler for the ADT~A08 message. That means that at the point when the applications message handler is executed by HLO, the routine already knows that the message is an ADT~A08. See PARSEA08^HLODEM5 in the section on Code Examples.

Example 13: In this example, the PID segment built in example 1 is parsed using the HLO segment parsing APIs. Note that each of the segment building APIs that inserts a data type into a segment has a corresponding API that parses the data type from the segment. See PID^HLODEM5 in the section on Code Examples.

Example 14: In this example, the NK1 segment built in example 3 is parsed using the HLO segment parsing APIs. See NK1^HLODEM5 in the section on Code Examples.

Example 15: In this example, the ADT~A08 message is parsed using hard-coded segment parsers. See PARSEA08^HLODEM6 in the section on Code Examples.

This example makes two WRONG assumptions:

1. That escape sequences do not appear in the message.
2. That the entire segment is stored on one node.

WARNING: With hardcoded segment parsers, the application is responsible for replacing escape sequences with the original characters. The application should NOT assume that the entire segment is contained in a single node.

3.5 Stepping through a Batch of Messages

A batch of messages is processed in a nearly identical manner to an individual message with these differences:

1. As with an individual message, the receiving application needs to be entered to the HLO APPLICATION REGISTRY, file # 779.2. However, the application needs to specify an application routine to execute when a batch of messages is received. It is up to that application routine to step through the batch of messages and determine each message's type and to process it accordingly.
2. The parsing of a batch starts off with a call to $$STARTMSG^HLOPRS, just as it does when parsing an individual message, but in addition the $$NEXTMSG^HLOPRS API is used to step through the batch of messages. Calling $$NEXTMSG^HLOPRS moves the parser to the next message within the batch.
3. If an application acknowledgment is required, the receiving application should return a batch of acknowledgments rather than an individual acknowledgment for each message within the original batch.

Example: Here is the general form for parsing a batch of messages. For simplicity, the steps needed to return application acknowledgments is not included here, but will be covered in a later section.

```
;{Create your workspace.}
N MSG,BHS,MSH,SEG
;
;{Start the parsing.}
I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.BHS) D Q
.{The message is lost or corrupted! Call an exception handler}
;
;{Step through the individual messages that are in the batch.}
F Q:'$$NEXTMSG^HLOPRS(.MSG,.MSH) D
.;
.:{If its not known in advance what type of messages the batch contains, then determine that from the message
header.}
.I MSH("MESSGE TYPE")="ADT",MSH("EVENT")="A08" D
..;
..;{Now step through the segments of the message.}
..F Q:'$$NEXTSEG^HLOPRS(.MSG,.SEG) D
…;
…;{Look at the segment type, then parse the needed data for that segment.}
…I SEG("SEGMENT TYPE")={3 character segment type} D
….;
….;{Use HLO's data type parsing routines to retrieve the data from the segment.}
..;
..;{Now that all the data has been retrieved from the message, process it.}
```

In this example:

- HLMSGIEN is the internal entry number of the message record in the HLO MESSAGES file (#778). If HLO is executing the application's routine in the background, then HLO will set that variable for the application's use.
- MSG is used by the HLO parser to keep track of the current message and segment.
- BHS will contain the fields that were parsed from the batch message header.
- Parsing always starts off with a call to $$STARTMSG^HLOPRS. It returns both the MSG array and the BHS array. If the function returns 0 then something is wrong – the message record has been deleted or is corrupted!
- To verify that this is a batch message rather than an individual message, the application may check MSG("BATCH")=1.
- After calling $$STARTMSG^HLOPRS, the HLO parser is pointing to the batch message header. Calling $$NEXTMSG^HLOPRS iteratively will step the parser through the messages within the batch.
- $$NEXTMSG^HLOPRS returns 1 if the parser has advanced to the next message. In that case, MSH will contain the individual fields parsed from the individual message header. The application can check the values of MSH("MESSAGE TYPE") and MSH("EVENT") to determine the type of message. HLO allows a batch of messages to contain mixed message types, though that is rarely done in practice.
- $$NEXTMSG^HLOPRS returns 0 if there are no more messages within the batch.
- After calling $$NEXTMSG^HLOPRS, the application then steps through that individual message's segments by calling $$NEXTSEG^HLOPRS, and parses that message in an identical manner to a message that is not contained within a batch.

Example 16
This is an example of parsing a batch of messages. The batch of messages consists of the ADT~A08 messages created in example 5. See BATCH^HLODEM7 in the appendix of code examples.

4.0 Acknowledgement Messages

HLO supports the two-phase enhanced acknowledgment protocol of the HL7 standard. The first phase is the exchange of a commit acknowledgment, also known as the accept acknowledgment. The second phase is the exchange of an application acknowledgment.

The sending application controls which acknowledgments to request. The information is encoded in the message header. For individual messages, the MSH-15 controls the Accept Acknowledgment Type and MSH-16 controls the Application Acknowledgment Type. *For batch messages, the standard does not provide a specific mechanism for requesting acknowledgments within the batch header. HLO encodes the information in a non-standard way using BHS-9.* As a result, batch messages cannot be exchanged between HLO and other messaging systems without customization.

The return of a commit acknowledgment signifies that the receiving system received the message and committed it to safe storage, and furthermore promises to deliver it to the receiving application.

NOTE: It is highly recommended that applications request the return of commit acknowledgments because otherwise there can be no reasonable certainty that a message will actually be delivered. Electronic communications can fail without warning!

The return of an application acknowledgment signifies that the receiving application processed the message. The distinguishing feature of an application acknowledgment is that it contains an MSA segment, not the message type or event. The MSA segment contains the message id of the original message and an acknowledgment code. The application acknowledgment message may contain segments with information related to the original message, such as the response to a query message. There is no general rule as to whether or not an application should use application acknowledgments, as its use is dictated by the specific requirements of the application.

4.1 Requesting Acknowledgments

The sending application indicates which acknowledgments to request at the point of calling one of the SEND API's listed in section 2.5, group 1.4. These parameters control which acknowledgments are requested:

PARMS("ACCEPT ACK TYPE")	=	"NE" means DO NOT request the commit ack.
	=	"AL" means DO request a commit ack.
	=	"" or left undefined – defaults to "AL"

If the application does request a commit acknowledgment, it may optionally indicate a callback routine as described in section 2.7.2. The routine will be executed by HLO when the commit acknowledgment is returned.

PARM("ACCEPT ACK RESPONSE")	=	<routine tag>^<routine>

Similarly, whether to request an application acknowledgment is controlled by this parameter:

PARMS("APP ACK TYPE")	=	"NE" means DO NOT request an application acknowledgment.
	=	"AL" means DO request an application acknowledgment.
	=	"" or undefined - defaults to "NE"

If the application does request an application acknowledgment, it may optionally indicate a callback routine as described in section 2.7.1. The routine will be executed by HLO when the application acknowledgment is returned. Although the use of a callback is optional, there is generally little point to requesting an application acknowledgment unless the application plans on taking some action upon its receipt.

PARMS("APP ACK RESPONSE")	=	<routine tag>^<routine>

Example: In this code snippet the application requests both a commit acknowledgment and an application acknowledgment. The application routine APPACK^HLODEM8 will be executed upon receipt of the application acknowledgment.

```
S PARMS("ACCEPT ACK TYPE")="AL"
S PARMS("APP ACK TYPE")="AL"
S PARMS("APP ACK RESPONSE")="APPACK^HLODEM8"
I '$$SENDONE^HLOAPI1(.MSG,.PARMS,.WHOTO,.ERROR) ;{perform exception processing on failure}
```

For application acknowledgments an alternative to setting up a callback routine as an input parameter is to register the application acknowledgment's message type and event in the HLO Application Registry (file #779.2). In either case, the result is identical – HLO will execute the application's routine when an application acknowledgment is received. At the point the application's routine is executed, the variable HLMSGIEN is set to the IEN of the application acknowledgment message. As always, the application should start processing the message by calling $$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.HDR), which in addition to returning the usual information will also return the identity of the original message as follows:

MSG("ACK TO")	=	the message id of the original message
MSG("ACK TO IEN")	=	the IEN of the original message in the HLO MESSAGES file (#778)

Example: As an alternative to adding a callback routine via an input parameter, as in the example above, the application could instead use the HLO Application Registry (file #779.2) to indicate what action to execute when an application acknowledgment is received. Here is how that could be accomplished:

```
APPLICATION NAME: HLO DEMO CLIENT
HL7 MESSAGE TYPE: ACK         HL7 EVENT: A01
 ACTION TAG: APPACK         ACTION ROUTINE: HLODEM8
 Package File Link: HL7 OPTIMIZED (HLO)
```

Example: This is an example of processing a returned application acknowledgment. In the case of an error, the application generates an alert to the support staff and reschedules the purge date of the original message so that there is plenty of time to correct the error. No action is taken if the acknowledgment does not return an error, but if this were, for example, a query result the application would process it. See APPACK^HLODEM8 in the appendix of code examples.

4.2 Returning Acknowledgments

4.2.1 Commit Acknowledgments

Commit acknowledgments are returned automatically by HLO with no action needed on the part of the receiving application. The commit acknowledgment consists of only two segments, the MSH and MSA. The message header is an MSH segment with message type of ACK with no event. The MSA segment contains these fields:

MSA-1	Acknowledgment Code = CA, CR, or CE.
MSA-2	Message ID of the original message
MSA-3	if applicable, a text message describing the error

The CA code (commit accept) indicates that the message was accepted by the receiving system and it will be passed to the receiving application. The CR (commit reject) and CE (commit error) codes indicate that some error condition was encountered and as a result the message will NOT be passed to the application.

4.2.2 Application Acknowledgments

It is the receiving application's responsibility to return an application acknowledgment if requested.

Individual Messages:
For an individual message, the receiving application may return an acknowledgment by following these steps:

1. As usual, $$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.HDR) is called to begin processing the original message.
2. If HDR("APPLICATION ACK TYPE")="AL" then the application must return an acknowledgment. If it is known in advance from the negotiated interface specification that an acknowledgment is required, then the test may be skipped.
3. Call $$ACK^HLOAPI2(.MSG,.PARMS,.ACK,.ERROR) to begin the process of creating the return message, where:
   a. MSG contains information about the original message, obtained in step 1.
   b. The PARMS array may contain these input parameters:

PARMS("ACK CODE")	- AA, AE, or AR (required)
PARMS("ERROR MESSAGE")	- text describing the error (optional)
PARMS("MESSAGE TYPE")	- message type of return message, defaults to "ACK"
PARMS("EVENT")	- event type of return message, defaults to the event of the original message.
PARMS("FIELD SEPARATOR")	- defaults to "|"
PARMS("ENCODING CHARACTERS")	– defaults to "^~\&"

   c. ACK is an array where the acknowledgment message is being constructed.
4. Optionally, the application may add segments to ACK in the usually manner. The application should not add a message header segment as that is always done automatically. The application may add an MSA segment, but if it does not, then one will be added automatically.
5. Call $$SENDACK^HLOAPI2 to complete the message and queue it for delivery.

Example: In this example, an application acknowledgment is returned. The message type defaults to ACK, the event defaults to the event of the original message, and the message delimiters default to the standard ones. The application chose to add an ERR segment to the acknowledgment message. The MSA segment with the AA code in MSA-1 and the message id of the original message in MSA-2 is automatically included in the message, since the application did not specifically add an MSA segment.

```
N MSH,HDR,ACK,SEG,PARMS
;
;start parsing the message
I '$$STARTMSG^HLOPRS(MSG,HLMSGIEN,.MSH) {report error} QUIT
;
;{finish parsing the message and process it}
;
;now test if an application acknowledgment was requested
I MSH("APP ACK TYPE")="AL" D
 .;
.;an acknowledgment was requested – so build one
 . S PARMS("ACK CODE")="AA"
 . I '$$ACK^HLOAPI2(.MSG,.PARMS,.ACK,.ERROR) {report error} QUIT
 .;
 . ;add an ERR segment
 . D SET^HLOAPI(.SEG,"ERR",0)
 . D SET^HLOAPI(.SEG,"0",3)
 . D SET^HLOAPI(.SEG,"I",4)
 . I '$$ADDSEG^HLOAPI(.ACK,.SEG) {report error} QUIT
 .;
 .;send the acknowledgment message – the MSA segment is automatically added
 . I '$$SENDACK^HLOAPI2(.ACK,.ERROR) { report error}
```

Example 17: This example is an extension of example 11. As in that example, this routine also receives and parses an ADT~A08 message, but in addition it returns an application acknowledgment. See PARSEA08^HLODEM10 in the appendix of code examples.

Batch Messages:
For a batch message, application acknowledgments for messages within the original batch are returned via a batch of messages by following these steps:

1. As usual, $$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.BHS) is called to begin processing the original message.
2. If MSG("BATCH")= 1 and HDR("APPLICATION ACK TYPE")="AL" then the application must return a batch of acknowledgments in response to this batch.
3. Call $$BATCHACK^HLOAPI2(.MSG,.PARMS,.ACK,.ERROR) to begin the process of creating the return batch, where:
   a. MSG contains information about the original message, obtained in step 1.
   b. The PARMS array may contain input parameters, including these:
      PARMS("FIELD SEPARATOR") - defaults to "|"
      PARMS("ENCODING CHARACTERS") – defaults to "^~\&"
   c. ACK is an array where the acknowledgment message is being constructed.

1. The receiving application should iteratively call $$NEXTMSG^HLOPRS to step through the messages of the original batch. After processing each message, the application may add an application acknowledgment to the return batch by calling $$ADDACK^HLOAPI3. Optionally, the application may also add other segments prior to calling $$ADDACK^HLOAPI3.

2. Call $$SENDACK^HLOAPI2 to complete the return batch of message acknowledgments and queue it for delivery.

Example: Returning a batch of acknowledgment in response to a batch.

```
N MSG,BHS,ACK,SEG,.ERROR
;{Start the parsing.}
I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.BHS) {report error} QUIT
.;
.;if not known in advance, check whether this is a batch message and if acknowledgments are requested
.;if yes, then start building the return batch in ACK
.I MSG("BATCH"),BHS("APP ACK TYPE")="AL" I '$$BATCHACK^HLOAPI3(.MSG,,.ACK,.ERROR) D
{report error} QUIT
.
;{Step through the individual messages that are in the batch.}
F Q:'$$NEXTMSG^HLOPRS(.MSG,.MSH) D
. ;
.{process each message}
.;
.if a return batch is being built, then add an application acknowledgment to the return batch
.S PARMS("APP ACK CODE")={"AA", "AE", or "AR"}
.I '$$ADDACK^HLOAPI3(.ACK,.PARMS,.ERROR) {report error} QUIT
.;
;now send the return batch
I '$$SENDACK^HLOAPI2(.ACK,.ERROR) {report error}
```

Example 18: This example is an extension of example 16. As in that example, this routine receives and parses a batch of messages, but in addition it returns a batch of application acknowledgments. See BATCH^HLODEM10 in the section on Code Examples.

4.2.3 Determining the Return Destination

Commit acknowledgments are always returned immediately over the same open connection as the original messages, so *where* to return the commit acknowledgment is never an issue.

But application acknowledgments are always returned at a later time over a different connection, so *where* to return the application acknowledgment message *is* an issue that needs to be addressed. HLO uses three methods to determine the return link in the HL LOGICAL LINK file (#870) in this order:

1. The application may optionally specify the return link as an input parameter to $$ACK^HLOAPI2 and $$BATCHACK^HLOAPI2. This method is recommended only if methods 2 and 3 are not applicable.
2. The entry for the receiving application in the HLO Application Registry (file #779.2) can specify a return link. This method is particularly valuable if the message needs to be returned via middleware, such as an interface engine, rather than directly.
3. If steps 1 and 2 don't provide the return link, HLO will try to determine it based on the message header of the original message. If the sending facility field contains a TCP/IP domain, or a VHA station number, HLO will perform a lookup on the HL LOGICAL LINK file (#779.2) using those values. If the sending facility of the original message is a VHA medical facility or national database with no middleware involvement, such as an interface engine, this is a reliable method for determining the return link.

Note: The Receiving Application and Receiving Facility for the application acknowledgment message header is always based on the Sending Application and Sending Facility of the original message header. The issue discussed above is slightly different – it concerns the determination of the IP address and port to connect with to return the application acknowledgment.

5.0 API Reference Guide

			Name	Brief Description	Section
Sending Messages					
			NEWMSG^HLOAPI	Begins a new individual message.	5.1.1
			NEWBATCH^HLOAPI	Begins a new batch of messages.	5.1.2
			ADDMSG^HLOAPI	Begins a new message within a batch of messages.	5.1.3
			ADDSEG^HLOAPI	Adds a segment to a message.	5.1.4
			MOVEMSG^HLOAPI	Moves a message built in the traditional style as an array of segments into an HLO message.	5.1.5
			MOVESEG^HLOAPI	Moves a segment built in the traditional style into the HLO message under construction.	5.1.6
	Setting Data into Segments				
			SET^HLOAPI	Sets a single atomic value into the segment. Use this if there is no specific API for the HL7 data type.	5.1.7.1
			SETAD^HLOAPI4	Sets an address.	5.1.7.2
			SETCE^HLOAPI4	Sets the CE data type (Coded Element).	5.1.7.3
			SETCNE^HLOAPI4	Sets the CNE data type (Coded Element with No Exceptions).	5.1.7.4
			SETCWE^HLOAPI4	Sets the CWE data type (Coded Value with Exceptions)	5.1.7.5
			SETDT^HLOAPI4	Sets a DT data type (Date).	5.1.7.6
			SETHD^HLOAPI4	Sets the HD data type (Hierarchic Designator)	5.1.7.7
			SETTS^HLOAPI4	Sets the TS data type (Timestamp)	5.1.7.8
			SETXPN^HLOAPI4	Sets the XPN data type (Extended Person Name)	5.1.7.9
	Addressing and Sending the Completed Message				
			SENDONE^HLOAPI1	Sends a message to a single recipient.	5.1.8.1
			SENDMANY^HLOAPI1	Sends a message to a list of recipients.	5.1.8.2
			SENDSUB^HLOAPI1	Sends a message to a list of recipients based on the HLO Subscription Registry.	5.1.8.3
			EN^HLOCNRT	Sends a message based on an Event/Subscriber Protocol setup.	5.1.8.4
Parsing Messages					
			STARTMSG^HLOPRS	Retrieves the message.	5.2.1
			NEXTMSG^HLOPRS	Advance to the next message within a batch	5.2.2
			NEXTSEG^HLOPRS	Advance to and parse the next segment	5.2.3
			HLNEXT^HLOMSG	Advances to the next segment without parsing it.	5.2.4
			MSGID^HLOPRS	Returns the message id from the message header.	5.2.5
	Parsing Data Types				
			GET^HLOPRS	Gets a value of unspecified data type.	5.2.6.1
			GETAD^HLOPRS2	Gets an AD data type (Address).	5.2.6.2

			GETCE^HLOPRS2	Gets a CE data type (Coded Element).	5.2.6.3
			GETCNE HLOPRS2	Gets a CNE data type (Coded Element With No Exceptions).	5.2.6.4
			GETCWE^HLOPRS2	Gets an CWE data type (Coded Element With Exceptions).	5.2.6.5
			GETDT^HLOPRS2	Gets a date.	5.2.6.6
			GETHD^HLOPRS2	Gets an HD data type (Hierarchic Designator)	5.2.6.7
			GETTS^HLOPRS2	Gets a timestamp.	5.2.6.8
			GETXPN^HLOPRS2	Gets an XPN data type (Extended Person Name)	5.2.6.9
Returning Application Acknowledgments					
			ACK^HLOAPI2	Begins an application acknowledgment for an individual message.	5.3.1
			BATCHACK^HLOAPI3	Begins an application acknowledgment for a batch of messages.	5.3.2
			ADDACK^HLOAPI3	Adds an acknowledgment message to a batch of acknowledgment messages.	5.3.3
			SENDACK^HLOAPI2	Sends the acknowledgment message.	5.3.4
Subscription Registry					
			CREATE^HLOASUB	Creates a new entry in the HLO SUBSCRIPTION REGISTRY file (#779.4).	5.4.1
			ADD^HLOASUB	Adds a new subscriber.	5.4.2
			END^HLOASUB	Terminates a subscriber.	5.4.3
			ONLIST^HLOASUB	Validates a subscriber.	5.4.4
			NEXT^HLOASUB	Loops through a subscriber list.	5.4.5
			INDEX^HLOASUB1	Creates a lookup value for an entry.	5.4.6
			FIND^HLOASUB1	Looks up an entry.	5.4.7
Miscellaneous APIs					
			RESEND^HLOAPI3	Queues a message for retransmission.	5.5.1
			REPROC^HLOAPI3	Queues a message for reprocessing.	5.5.2
			PROCNOW^HLOAPI3	Reprocesses a message immediately.	5.5.3
			SETPURGE^HLOAPI3	Resets the scheduled purge time for a message.	5.5.4
			QUECNT^HLOQUE	Returns a count of messages pending on each queue of all types.	5.5.5
			OUT^HLOQUE	Returns a count of messages pending on each outgoing queue.	5.5.6
			IN^HLOQUE	Returns a count of messages pending on each incoming queue.	5.5.7
			SEQ^HLOQUE	Returns a count of messages pending on each sequence queue.	5.5.8

5.1 Sending Messages

5.1.1 Start Building a New Message

<u>Routine:</u> $$NEWMSG^HLOAPI(.PARMS,.MSG,.ERROR)

<u>Description:</u> Starts the process of building an individual messag.

Input:

PARMS These subscripts are used:		Required	Used to pass in various parameters.
	“COUNTRY”	Optional	A three-character country code.
	“CONTINUATION POINTER”	Optional	Indicates a fragmented message.
	“EVENT”	Required	A three-character event type.
	“FIELD SEPARATOR”	Optional	Field separator that defaults to "|".
	“ENCODING CHARACTERS”	Optional	Four HL7 encoding characters that defaults to "^~\&".
	“MESSAGE STRUCTURE”	Optional	MSH-9, component 3 - a code from the standard HL7 table.
	“MESSAGE TYPE”	Required	A three-character message type (required).
	“PROCESSING MODE”	Optional	MSH-11, component 2 – A one character code.
	“VERSION”	Optional	The HL7 Version ID, for example, "2.4", defaults to 2.4.

Output: Function call - Returns 1 on success, 0 on failure.

MSG	Required	Pass-by-Reference	Used by the HLO as a workspace to build the message.
ERROR	Optional	Pass-by-Reference	Returns an error message on failure.

5.1.2 Start Building a New Batch of Messages

Routine: $$NEWBATCH^HLOAPI(.PARMS,.MSG,.ERROR)

Description: Starts the process of building a batch of messages.

Input:

PARMS These subscripts are used:		Optional	Used to pass in various parameters.
	“COUNTRY”	Optional	A three-character country code.
	“FIELD SEPARATOR”	Optional	Field separator that defaults to "|".
	“ENCODING CHARACTERS”	Optional	Four HL7 encoding characters that defaults to "^~\&".
	“VERSION”	Optional	HL7 Version ID, for example, "2.4" that defaults to 2.4.

Output: Function call - returns 1 on success, 0 on failure.

MSG	Required	Pass-by-Reference	Used by HLO as a workspace for building the message.
ERROR	Optional	Pass-by-Reference	Returns an error message on failure.

5.1.3 Add a New Message to a Batch

Routine: $$ADDMSG^HLOAPI(.MSG,.PARMS,.ERROR)

Description: Add a message to a batch that is in the process of being built.

Input:

MSG		Required	Pass-by-Reference	Used by HLO as a workspace for building the message.
PARMS These subscripts are used:		Required	Pass-by-Reference	Used to pass in various parameters.
	"EVENT"	Required	A three-character event type for the message being added.	
	"MESSAGE TYPE"	Required	A three-character message type for the message being added.	

Output: Function Call - Returns 1 on success, 0 on failure.

MSG	Required	Pass-by-Reference	Used by HLO as a workspace for building the message.
ERROR	Optional	Pass-by-Reference	Returns an error message on failure.

5.1.4 Add a Segment to a Message

Routine: $$ADDSEG^HLOAPI(.MSG,.SEG,.ERROR,TOARY)

Description: Used to add a segment to a message that is being built.

Input:

MSG	Required	Pass-by-Reference	Used by HLO as a workspace for building the message.
SEG	Required	Pass-by-Reference	Contains the segment's data. It must be built calling the APIs in section 5.1.7 prior to calling ADDSEG^HLOAPI.

Note#1: The message control segments, including the MSH and BHS segments, are added automatically.
Note#2: Prior to calling ADDSET^HLOAPI, at a minimum the three- character segment type must have been set into SEG.
Note#3: SEG is killed upon successfully adding the segment.

Output: Function call, returns 1 on success, 0 on failure.

MSG	Required	Pass-by-Reference	Used by HLO as a workspace for building the message.
ERROR	Optional	Pass-by-Reference	Returns an error message on failure.
TOARY	Optional	Pass-by-Reference	Returns the built segment as a set of lines in the format: TOARY(1) TOARY(2) TOARY(3), etc. If the segment fits on a single node then the entire segment is returned in TOARY(1).

5.1.5 Move Traditional-Style Message Array into an HLO Message

Routine: MOVEMSG^HLOAPI(.MSG,.ARY)

Description: If a message is built as an array of segments as done in HL7 1.6 , this API can be used to move the message array into the MSG workspace. This API allows message builders that were created prior to HLO to be used within HLO.

Input:

MSG	Required	Pass-by-Referer	Used by HLO as a workspace for building the message.
ARY	Required	Pass-by-Value	The name of the local or global array where the message was built.

Output:

MSG	Required	Pass-by-Reference	Used by HLO as a workspace for building the message. MSG() is updated with the contents of @ARY@().

NOTE: When using this API, it is the application's responsibility to insure that message delimiters that may occur within the data have been replaced with the corresponding escape sequence.

Example: If a message (excluding the header) was built as an array of segments stored in ARRAY(1), ARRAY(2), etc., then call MOVEMSG^HLOAPI:

```
        S ARY="ARRAY"
        D MOVEMSG^HLOAPI(.MSG,ARY)
```

5.1.6 Move Traditional-Style Segment Array into HLO

Routine: $$MOVESEG^HLOAPI(.MSG,.SEG,.ERROR)

Description: Used to add a segment built in the old-style of HL7 1.6 into an HLO message that is being built. Its main use is to allow segment builders that were designed for use with HL7 1.6 messages to be used with HLO.

Input:

MSG	Required	Pass-by-Reference	Used by HLO as a workspace for building the message.
SEG	Required	Pass-by-Reference	Contains the segment. If the segment fits on a single node then SEG is the entire segment. If it does not fit on a single node then the annex nodes are (SEG(1), SEG(2), etc.

Output: Function call, returns 1 on success, 0 on failure.

MSG	Required	Pass-by-Reference	Used by HLO as a workspace for building the message. After calling $$MOVESEG^HLOAPI the segment stored in SEG is appended to the message being built in MSG.
ERROR	Optional	Pass-by-Reference	Returns an error message on failure.

NOTE: When using this API, it is the application's responsibility to insure that message delimiters that may occur within the data have been replaced with the corresponding escape sequence.

5.1.7 Setting Data Types into a Segment

The HL7 standard includes numerous data types, such as for dates, addresses, and coded values. HLO provides specialized APIs for setting some of the most common data types into a segment, but not all of them. If HLO lacks a specialized API for the data type at hand, the application should use SET^HLOAPI to set the data directly into the segment. If the data type has multiple parts, then call SET^HLOAPI multiple times to set each part into the segment.

HLO will automatically replace message delimiters found within the data with the corresponding escape sequence.

Implementation Note: Calling these APIs sets the data into the SEG array, which a workspace used by HLO for building the segment. The format of this internal workspace is:

SEG(<field sequence #>,<occurrence # of repetition>,<component #>,<sub-component #>)=<sub-component value>

When $$ADDSEG^HLOAPI is called HLO will concatenate all the values together and insert the message delimiters to create the completed segment. The segment is then appended to the message array that is being built.

Both the segment array and the message array are internal to HLO and should not be written to directly by the application.

5.1.7.1 Set Value (Unspecified Data Type)

Routine: SET^HLOAPI(.SEG,VALUE,FIELD,COMP,SUBCOMP,REP)

Description: Sets a single atomic value of unspecified data type into a segment at a specified location. It is used where HLO lacks a specific API for the data type at hand.

Input:

SEG	Required	Pass-by-Reference	The segment being built.
VALUE	Required	Pass-by-Value	The individual value to be set into the segment.
FIELD	Optional	Pass-by-Value	The sequence number of the field (optional, defaults to 0). **Note:** FIELD=0 is used to denote the segment type.
COMP	Optional	Pass-by-Value	The number of the component (optional, defaults to 1).
SUBCOMP	Optional	Pass-by-Value	The number of the subcomponent that defaults to 1.
REP	Optional	Pass-by-Value	The occurrence number , which defaults to 1. For a non-repeating field, the occurrence number need not be provided, because it would be 1.

Output:

SEG	Required	Pass-by-Reference	An array that HLO uses as its private workspace while building a segment.

Example: Sets the segment type to MSA.

```
D SET^HLOAPI(.SEG,"MSA",0)
```

Example: Sets the value “AE” into the first field. It defaults to the first occurrence of the field, first component, and first sub-component.

```
D SET^HLOAPI(.SEG,"AE",1)
```

Example: Sets the value “ALBANY” into the first field, fourth occurrence of a repeating field, second component, third sub-component.

```
D SET^HLOAPI(.SEG,"ALABANY",1,2,3,4)
```

5.1.7.2 Set Address

Routine: SETAD^HLOAPI4(.SEG,.VALUE,FIELD,COMP,REP)

Description: Sets an AD data type (Address) into the segment at the specified location. It can also be used to set the 1st 8 components of the XAD (Extended Address) data type.

Note: The XAD (extended address) data type replaced the AD data type as of version 2.3 of the HL7 standard. The XAD data type has additional components not included in the AD data type. If the additional components of the XAD data type are needed, SETAD^HLOAPI can be called, then the additional parts set into the segment by calling SET^HLOAPI4.

Input:

SEG	Required	Pass-by-Reference	The segment being built.
VALUE	Required	Pass-by-Reference	These subscripts may be passed: • "STREET1" - street address • "STREET2" - other designation • "CITY" • "STATE" - state or province • "ZIP" - zip or postal code • "COUNTRY" • "TYPE" - address type
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the address is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

SEG	Required	Pass-by-Reference	The segment being built.

Example: Sets an address into the second field of the segment. Since a component is not specified, its parts are at the component level.

```
S VALUE(“STREET1”)=”13 MOCKING BIRD LANE”
S VALUE(“CITY”)=”ALBANY”
S VALUE(“STATE”)=”NY”
S VALUE(“ZIP”)=”12506”
S VALUE(“TYPE”)=”H”
D SETAD^HLOAPI4(.SEG,.VALUE,2)
```

Example: Sets an address into the second field of the segment. Since a component is specified, the address is demoted to the component level, and its parts are at the subcomponent level.

```
S VALUE(“STREET1”)=”13 MOCKING BIRD LANE”
S VALUE(“CITY”)=”ALBANY”
S VALUE(“STATE”)=”NY”
S VALUE(“ZIP”)=”12506”
S VALUE(“TYPE”)=”H”
D SETAD^HLOAPI4(.SEG,.VALUE,2,1)
```

5.1.7.3 Set Coded Element

Routine: SETCE^HLOAPI4(.SEG,.VALUE,FIELD,COMP,REP)

Description: Sets the CE data type (Coded Element) into the segment at the specified location.

Input:

SEG	Required	Pass-by-Reference	The segment being built.
VALUE	Required	Pass-by- Reference	These subscripts may be passed: "ID" - the identifier "TEXT" - "SYSTEM" - name of the code system "ALTERNATE ID" - alternate identifier "ALTERNATE TEXT" "ALTERNATE SYSTEM" - name of the alternate coding system
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the coded element is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the

			field.
REP	Optional	Pass-by-Value	The occurrence number, defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

SEG	Required	Pass-by-Reference	The segment being built.

5.1.7.4 Set Coded Element with No Exceptions

Routine: SETCNE^HLOAPI4(.SEG,.VALUE,FIELD,COMP,REP)

Description: Sets the CNE data type (Coded Element with No Exceptions) into the segment at the specified location.

Input:

SEG	Required	Pass-by-Reference	The segment being built.
VALUE	Required	Pass-by- Reference	These subscripts may be passed: • "ID" - the identifier • "TEXT" • "SYSTEM" - name of the code system • "ALTERNATE ID" - alternate identifier • "ALTERNATE TEXT" • "ALTERNATE SYSTEM" - name of the alternate coding system • "ORIGINAL TEXT"
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the coded element is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields this parameter is not necessary.

Output:

SEG	Required	Pass-by-Reference	The segment being built.

5.1.7.5 Set Coded Value with Exceptions

Routine: SETCWE^HLOAPI4(.SEG,.VALUE,FIELD,COMP,REP)

Description: Sets the CWE data type (Coded Value with Exceptions) into a segment at the specified location.

Input:

SEG	Required	Pass-by-Reference	The segment being built.
VALUE	Required	Pass-by-Reference	These subscripts may be passed: • "ID" - the identifier • "TEXT" • "SYSTEM" - name of the code system • "ALTERNATE ID" - alternate identifier • "ALTERNATE TEXT" • "ALTERNATE SYSTEM" - name of the alternate coding system • "ORIGINAL TEXT"
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the coded value is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

SEG	Required	Pass-by-Reference	The segment being built.

5.1.7.6 Set Date

Routine: SETDT^HLOAPI4(.SEG,.VALUE,FIELD,COMP,REP)

Description: Sets a DT data type (Date) in FileMan format into the segment at the specified location. The date is converted to HL7 format. The degree of precision may be optionally specified.

Input:

SEG	Required	Pass-by-Reference	The segment being built.
VALUE	Required	Pass-by-Reference	The date to be set into the segment. Optionally, you may specify that the value should be rounded down to a particular precision by specifying this subscript: "PRECISION" (If needed, VALUE must be passed by reference.) Allowed values are: • "D" - day (default value) • "L" - month • "Y" - year
FIELD	Required	Pass-by-Value	The sequence # of the field.
COMP	Optional	Pass-by-Value	If specified, the date is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence #, defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

SEG	Required	Pass-by-Reference	The segment being built.

5.1.7.7 Set Hierarchic Designator

Routine SETHD^HLOAPI4(.SEG,.VALUE,FIELD,COMP,REP)

Description: Sets the HD data type (Hierarchic Designator) into the segment at the specified location.

Input:

SEG	Required	Pass-by-Reference	The segment being built.
VALUE	Required	Pass-by- Reference	These subscripts may be passed: • "NAMESPACE ID" • "UNIVERSAL ID" • "UNIVERSAL ID TYPE"
FIELD	Required	Pass-by-Value	The sequence # of the field.
COMP	Optional	Pass-by-Value	If specified, the hierarchic designator is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

SEG	Required	Pass-by-Reference	The segment being built.

5.1.7.8 Set Timestamp

Routine: SETTS^HLOAPI4(.SEG,.VALUE,FIELD,COMP,REP)

Description: Sets the TS data type (Timestamp) in FM format into a segment at the specified location. The time stamp is converted to HL7 format. The degree of precision may be optionally specified. The inserted value will include the time zone if the input included the time

Input:

SEG	Required	Pass-by-Reference	The segment being built.
VALUE	Required	If the precision is not needed, then Pass-by-Value may be used. If the precision is specified, then Pass-by-Reference.	The date/time in FileMan format. You can *optionally* specify that the value is to be rounded down to a particular precision by specifying this subscript: "PRECISION" - Allowed values are: • "S" - second • "M" - minute • "H" - hour • "D" - day • "L" - month • "Y" - year • "" - precision not specified
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the timestamp is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fiel this parameter is not necessary.

Output:

SEG	Required	Pass-by-Reference	The segment being built.

5.1.7.9 Set Extended Person Name

Routine: SETXPN^HLOAPI4(.SEG,.VALUE,FIELD,COMP,REP)

Description: Sets the XPN data type (Extended Person Name) into the segment at the specified location.

Note: The XPN data type has additional components not included in this API. If needed, the additional components can be set into the segment by calling SET^HLOAPI after calling SETXPN^HLOAPI4.

This API can also be used to set the Person Name (PN) data type into a segment. The PN data type was made obsolete in version 2.5 of the HL7 standard.

Input:

SEG	Required	Pass-by-Reference	The segment that is being built.
VALUE	Required		These subscripts may be passed: • "FAMILY" • "GIVEN" first name • "SECOND" second and further names or initials • "SUFFIX" (e.g., JR) • "PREFIX" (e.g., DR) • "DEGREE" (e.g., MD)
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the extended person name is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the fiel
REP	Optional	Pass-by-Value	The occurrence number, defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

SEG	Required	Pass-by-Reference	The segment being built.

5.1.8 Completing and Sending Messages

Once the building of the message has been completed with all its segments, it must be addressed and sent. HLO supports several APIs for doing that. They are:

SENDONE^HLOAPI1	Sends a single copy of the message to a single destination.
SENDMANY^HLOAPI1	Sends one or many copies of the message to a list of destinations. The list of destinations is provided as an input parameter to the API. This API is suited to static lists of recipients because the list is created within the routine and cannot be modified without a patch.
SENDSUB^HLOAPI1	Sends one or many copies of the message to a list of subscribers. The list of subscribers is maintained in an entry in the HLO Subscription Registry. This API is suited to applications that need to provide for a dynamic list of subscribers. Subscribers can be added, modified, or deleted via the table-based HLO Subscription Registry.
EN^HLOCNRT	This API is supports existing applications based on HL7 1.6 that need to convert to HLO. It is not recommended for new applications. It supports many of the features of GENERATE^HLMA, though not all of

them. It requires that the message be passed in as an array of segments, and uses the traditional-style Event/Subscriber protocol setup.

5.1.8.1 Send a Single Message

Routine: $$SENDONE^HLOAPI1(.MSG,.PARMS,.WHOTO,.ERROR)

Description: Used to send a message to a single recipient. The recipient is identified in the message header by the Receiving Facility and the Receiving Application. The message may optionally be routed through middleware.

Input:

MSG	Required	Pass-by-Reference	The workspace where the message was built.
PARMS These subscripts are allowed:	Required	Pass-by-Reference	Various parameters.
"APP ACK RESPONSE"	Optional		<tag^routine> to call in response to app ack. (see Callbacks section 2.7)
"APP ACK TYPE"	Optional		<AL,NE> defaults to NE.
"ACCEPT ACK RESPONSE"	Optional		<tag^routine> to call in response to a commit ack. (See Callbacks section 2.7.)
"ACCEPT ACK TYPE"	Optional		<AL,NE> defaults to AL.
"FAILURE RESPONSE"	Optional		<tag^routine> The sending application routine to execute when the transmission of the message fails, i.e., the message cannot be sent or no commit ack is received. (See Callbacks section 2.7.)
"QUEUE"	Optional		An application can name its own private queue by entering a string under 20 characters. The queue name should be namespaced.
"SECURITY”	Optional		Security information to include in the header segment, MSH-8.
"SENDING APPLICATION"	Required		The name of the sending application (60 characters max-length).
“SEQUENCE QUEUE”	Optional		A sequence queue to place the message on, name up to 30 characters and must be namespaced. See section 2.8 Sequence Queues.
WHOTO These subscripts are allowed:	Required	Pass-by-Reference	Used to specify the receiving application, receiving facility, and HL Logical Link over which to transmit.
"RECEIVING APPLICATION"	Required		The name of the application that should receive the message (60 characters max-length).
One of the following four parameters is required to identify the Receiving Facility.			
"FACILITY LINK IEN"	Optional		IEN of the logical link.

	"FACILITY LINK NAME"	Optional	Name of the logical link.
	"INSTITUTION IEN"	Optional	Pointer to the INSTITUTION File (#4).
	"STATION NUMBER"	Optional	Station number with suffix.
	One of the following two parameters MAY be provided, optionally, to identify the interface engine or other middleware to route the message through.		
	MIDDLEWARE LINK IEN"	Optional	Pointer to a logical link for the interface engine or other middleware.
	"MIDDLEWARE LINK NAME"	Optional	Name of the logical link for the interface engine or other middleware.

Output: Function call - returns the IEN of the message in HLO MESSAGES file (#778) on success, 0 on failure.

MSG	Required	Pass-by-Reference	The workspace where the message was built.
ERROR	Optional	Pass-by-Reference	On failure, will contain an error message.
PARMS		Pass-by-Reference	Killed when the function returns.
WHOTO		Pass-by-Reference	Killed when the function returns.

5.1.8.2 Send Messages to a List of Destinations

Routine: $$SENDMANY^HLOAPI1(.MSG,.PARMS,.WHOTO)

Description: Used to send a message to a list of recipients. The list is passed in as an array.

Input: Similar to $$SENDONE^HLOAPI1. In $$SENDMANY^HLOAPI1, the WHOTO() array is a list of recipients.

MSG	Required	Pass-by-Reference	The workspace where the message was built.
PARMS	Required	Pass-by-Reference	See PARMS definition in the $$SENDONE^HLOAPI1 API.
WHOTO	Required	Pass-by-Reference	Specifies a list of recipients. Each recipient should be listed individually in array WHOTO(i), where i=a recipient. For each recipient, the same subscripts may be defined as in the $$SENDONE API. For example: WHOTO(1,"FACILITY LINK NAME")="VAALB" WHOTO(1,"RECEIVING APPLICATION")="MPI" WHOTO(2,"STATION NUMBER")=500 WHOTO(2,"RECEIVING APPLICATION")="MPI"

Output: Function call- returns 1 if a message is queued to be sent to each intended recipient; 0 otherwise.

MSG	Required	Pass-by-Referen	The workspace where the message was built.
PARMS	Required	Pass-by-Referen	Killed when the function returns.
WHOTO	Required	Pass-by-Referen	Returns the status of each message to be sent in the format: (<i>,"QUEUED") - 1 if queued to be sent, 0 otherwise. (<i>,"IEN") – IEN from HLO MESSAGES file (#778) if queued to be sent, null otherwise. (<i>,"ERROR") - Error message if an error was encountered (status=0), and null otherwise.

5.1.8.3 Send Messages via the HLO Subscription Registry to a List of Subscribers

Routine: $$SENDSUB^HLOAPI1(.MSG,.PARMS,.MESSAGES)

Description: Used to send a message to a list of recipients. It differs from $$SENDMANY^HLOAPI1 in that the list, instead of being passed in as an array, is contained in the HLO SUBSCRIPTION REGISTRY file (#779.4). In other words, the list of recipients is maintained by the application in a table rather than being created dynamically.

Input:

MSG	Required	Pass-by-Reference	The workspace where the message was built.
PARMS	Required	Pass-by-Reference	See PARMS definition in the $$SENDONE^HLOAPI1 API. It has one additional subscript, PARMS(“SUBSCRIPTION IEN”).
PARMS("SUBSCRIPTION IEN")	Required	Pass-by-Reference	The IEN of an entry in the HLO. SUBSCRIPTION REGISTRY File (#779.4), defining the intended recipients of this message.

Output: Function call - returns 1 if a message is queued to be sent to each intended recipient, 0 otherwise.

MSG	Required	Pass-by-Reference	The workspace where the message was built.
PARMS			Killed when the function returns.
MESSAGES	Required	Pass-by-Reference	Returns the status of each message to be sent in this format, where the sub-IEN is the IEN of the recipient in the RECIPIENTS sub-file of the HLO SUBSCRIPTION REGISTRY File (#779.4). (<subien>,"QUEUED") - 1 if queued to be sent, 0 otherwise. (<subien>,"IEN") – IEN from HLO MESSAGES file (#778) if queued to be sent, or null otherwise. (<subien>,"ERROR") - Error message if an error was encountered (status=0), and null otherwise.

5.1.8.4 Send Messages via HL7 1.6 Protocol Setup

Routine: $$EN^HLOCNRT(HLOPRTCL,.ARYTYP,.HLP,.HLL, .RESULT)

Description: Used to send a message that was built in the traditional-style of HL 1.6 and send it via an Event/Subscriber protocol setup as used in HL7 1.6. It is similar to GENERATE^HLMA, though it does not implement all of its features.

Protocol Fields Not Supported:

- Entry Action
- Exit Action
- Processing ID
- Response Processing Routine
- Sending Facility Required
- Receiving Facility Required
- Processing Routine
- Routing Logic

Input:

HLOPRTCL	Required	Pass-by-Value	Protocol IEN or Protocol Name
ARYTYP	Required	Pass-by-Value	An array type: “GM” is used for a global array and “LM” is used for a local array.
HLP These subscripts are allowed:	Optional	Pass-by-Referen	Used to pass in various parameters.
“SECURITY”	Optional		Security information to include in the header segment, SEQ-8
“CONTPTR”	Optional		Value to place in MSH-14, the Continuation Pointer field used to link long messages that have been broken into fragments. Its use is application-specific and not defined by the standard.
“QUEUE”	Optional		An application can name its own private queue by entering a string under 20 characters. The queue name should be namespaced.
“SEQUENCE QUEUE”	Optional		A sequence queue to place the message on, name up to 30 characters and must be namespaced. See section 2.8, Sequence Queues.
“EXCLUDE SUBSCRIBER”	Optional		Used to specify subscriber protocols that should be skipped when sending the HL7 message.
HLL	Optional	Pass-by-Reference	The HLL array is used by some applications to dynamically address messages by passing in a list of recipients. The format of the list is: HLL(“LINKS”,<i=1,2,3,etc.>)=<destination protocol, name or IEN>^<destination link, name or IEN>

Output: Function Call - returns 1 on success, 0^error code^error description on failure.

RESULT	Required	Pass-by-Reference	**Note:** If more than one message was sent, the RESULT applies to the first. Status information of the additional messages located at RESULT(1), Result(2), etc. **On success:** <subscriber protocol IEN>^<link IEN>^<message id>^0 **On failure:** <subscriber protocol IEN>^<link IEN>^<message id>^<error code>^<optional error message>
RESULT("IEN")	Optional	Pass-by-reference	The IEN, file 778, of the first message sent.
RESULT(<1,2,…>)	Optional	Pass-by-Reference	Status information for additional messages sent. The returned status information is in the same format as for RESULT above.
RESULT(<1,2,…>, "IEN")	Optional	Pass-by-Reference	The IENs, file 778, of additional messages sent.

5.2 Parsing Messages

5.2.1 Start Parsing a Message

Routine: $$STARTMSG^HLOPRS(.MSG,.HLMSGIEN,.HDR)

Description: This API is always called as the first step in parsing a message. It retrieves the message header and parses it. It also returns administrative information about the message.

The application will most often parse a message while it is executing in the context of the background HLO process that passes newly-received messages that are pending on the incoming queue to the application. At that point, the variable HLMSGIEN is guaranteed to be defined and its value set to the IEN of the newly received message.

Input:

HLMSGIEN	Required	Pass-by-Value	The IEN of the message in HLO MESSAGES file (#778).

Output: Function returns 1 on success, 0 on failure. Failure would indicate that the message was not found.

MSG	Required	Pass-by-Reference	This array is used by HLO as a workspace to retrieve the message. The application MUST NOT write to this array.
The following subscripts are returned and may be referenced by the application:			
"ACK BY"	The message ID of the message that acknowledges this one.		
* "ACK BY IEN"			
"ACK TO"	The message ID of the message that this message acknowledges.		

* "ACK TO IEN"	The message IEN (file #778) of message that this one acknowledges. If the message is an individual message within a batch, the value is <IEN>^<sub-IEN>, where sub-IEN is the IEN of the message within the sub-file #778.03.
"BATCH"	A flag that is set to 1 if the message is a batch message and 0 if it is not a batch message.
"BODY"	The IEN of the entry in file #777 that contains the body of the message.
"DIRECTION"	Indicates the direction of the message transmission, "IN" if incoming, "OUT" if outgoing.
"DT/TM"	The date/time that the message was sent or received.
"DT/TM CREATED"	The date/time the message record was created.
* "EVENT"	The HL7 event, only defined if not a batch message.
"HDR"	The message header segment, NOT parsed. MSG(“HDR”,1) contains fields sequence 1-6, and MSG(“HDR”,2) contains fields sequence 7-end.
"ID"	The id from within the message header. It is the Message Control ID field for an individual message and the Batch Control ID field for a batch message
"IEN"	The IEN of the message in the HLO MESSAGES file (#778).
*"MESSAGE TYPE"	The HL7 message type, only defined if not a batch message.
"STATUS"	The message’s completion status. It is NULL if the message transaction has not yet been completed, “SU” if the message was completed without an error returned, “ER” if transmission failed or an error was returned.
	These subscripts are stored at a lower level under “STATUS”:

	"ACCEPT ACK'D"	Indicates whether an accept acknowledgment, aka commit acknowledgment, was sent or received. Its value is 1 yes, 0 if no.
	* ”ACCEPT ACK DT/TM”	If a commit acknowledgment was sent or received, this is the date/time.
	* ”ACCEPT ACK ID”	If a commit acknowledgment was sent or received, this is its message id.
	* “ACCEPT ACK MSA”	If a commit acknowledgment was sent or received, this is the entire MSA segment that it contained.
	"APP ACK'D"	Indicates whether an application acknowledgment was sent or received. Its value is 1 yes, 0 if no.
	“ERROR TEXT"	If the completion status is “ER”, this is any text that is associated with that status. If the status is an error because of an acknowledgment that was returned, it’s the text that was returned in the MSA segment.

	"LINK NAME"	The name of the link in the HL LOGICAL LINK file (#870) over which the message was transmitted or will be transmitted.
	"PURGE"	If the message has been scheduled for purging, this is the date/time of the scheduled purge.
	* "SEQUENCE QUEUE"	If this is an outgoing message that was initially placed on a sequence queue, this is the name of the sequence queue.

* Indicates that the subscript is not always defined. Some subscripts are defined only if they are valued, ergo, for a batch message the subscript "MESSAGE TYPE" is not defined.

	HDR	Optional	Pass-by-Reference	This array contains the results of parsing the message header. Escape sequences are replaced by the original characters. See section 3.3 for the subscripts returned (HLO Parses the Message Header).

5.2.2 Advance to the Next Message within a Batch

Routine: $$NEXTMSG^HLOPRS(.MSG,.MSH)

Description: Used to advance to the next message within a batch and return the MSH segment for that message.

Input:

MSG	Required	Pass-by-Reference	This array is used to track the current position within the message.

Output: Function returns 1 on success, 0 if there are no more messages.

MSG	Required	Passed by Reference	Updated with the new position within the message.
MSH	Required	Pass-by-Reference	Returns the parsed message header. See section 3.3.1 for the subscripts returned (Parsing the MSH Segment)

5.2.3 Advance to and Parse the Next Segment

Routine: $$NEXTSEG^HLOPRS(.MSG,.SEG)

Description: Used to advance parsing to the next segment and parses it.

Input:

MSG	Required	Pass-by-Reference	This array is used to track the current position in the message.

Output: Function Call- returns 1 on success, 0 if there are no more segments in this message. For batch messages, a return value of 0 does not preclude the possibility that there are additional individual messages within the batch.

MSG	Required	Pass-by-Reference	Updated with the new position within the message.
SEG	Required	Pass-by-Reference	Contains all the individual values parsed from the segment. The application should not reference this array directly. Instead, the APIs listed in section 5.2.4 below should be used. Additionally, SEG("SEGMENT TYPE") will return the 3 character segment type that starts every segment. SEG(0) is also returned and is shorthand for SEG("SEGMENT TYPE") with the same return value.

5.2.4 Get Next Segment

Routine: $$HLNEXT^HLOMSG(.MSG,.SEG)

Description:

This API returns the next segment in the current message, but unlike $$NEXTSEG^HLOPRS, it does not parse it. If the message is within a batch, it will NOT jump to the next message when reaching the last segment in the current message.

Its use is principally for messaging applications that were developed prior to HLO that need to convert to HLO. It steps through messages in a similar fashion to HL7 1.6's 'XECUTE HLNEXT', with two differences:

1. For batch messages, $$HLNEXT does not traverse to the next message in the batch as 'X HLNEXT' does. Instead, too step to the next message within a batch the application must call $$NEXTMSG^HLOPRS.
2. $$HLNEXT^HLOMSG always returns the data in an array in the format SEG(1), SEG(2), etc. The number of subscripts used depends on how large the segment is. 'XECUTE HLNEXT' instead returns the variable SEG containing the segment, with SEG(1), SEG(2), etc. defined only if the segment doesn't fit entirely in SEG.

Input:

MSG	Required	Pass-by-Reference	Contains the message and tracks the current position within the message.

Output: Function returns 1 on success, 0 if there are no more segments in this message. For batch messages, a return value of 0 does not preclude the possibility that there are additional individual messages within the batch.

MSG	Required	Pass-by-Reference	Updated with the current position within the message.

SEG	Required	Pass-by-Reference	The segment is returned in this array in the format SEG(1), SEG(2), etc. If the segment fits on a single node, only SEG(1) is returned.

Note: After calling this API, it is the applications responsibility to correctly parse the individual fields, including:

- Converting data types
- Replacing escape sequences with the original characters
- Where the segment is returned in more than one node with fields broken in parts, correctly putting together the pieces.

5.2.5 Get Message ID

Routine: $$MSGID^HLOPRS(HLMSGIEN)

Description: Given the message IEN (file #778) this function returns the message ID from the message header.

Input:

HLMSGIEN	Required	Pass-by-Value	The message IEN, from the HLO MESSAGES file (#778).

Output: The function returns the message ID from the message header.

5.2.6 Parsing Data Types

After calling $$NEXTSEG^HLOPRS, the segment has been totally parsed and all its data returned in an array. The following APIs are used to fetch the specific data that is needed from that array.

The HLO standard defines numerous data types, such as for person names, addresses, and coded values. HLO provides specialized APIs to get some of the HL7 data types, but not all of them. Where a specialized API is not available, the generic $$GET^HLOPRS should be used to get the needed data. It returns a single atomic value, so if the data has more than one part, then $$GET^HLOPRS should be called multiple time to obtain the entire data type.

The following APIs are used to obtain the specific data that is needed after the segment has been parsed by $$NEXTSEG^HLOPRS.

5.2.6.1 Get Value (Unspecified Data Type)

Routine: $$GET^HLOPRS(.SEG,FIELD,COMP,SUBCOMP,REP)

Description: Gets a single atomic value from a segment. It is used when HLO lacks an API specific to the data type at hand.

Example: $$GET^HLOPRS(.SEG,1) specifies to return a value of the first field. Since the other parameters aren't specified it defaults to the first component, first subcomponent, first occurrence.

Input:

SEG	Required	Pass-by-Reference	The parsed segment.
FIELD	Optional	Pass-by-Value	The sequence number of the field which defaults to 1. If 0 (zero) is specified, then the function returns the segment type.
COMP	Optional	Pass-by-Value	The number of the component which defaults to 1.
SUBCOMP	Optional	Pass-by-Value	The number of the subcomponent which defaults to 1.
REP	Optional	Pass-by-Value	The occurrence number which defaults to 1. For a non-repeating field, the occurrence number will always be 1.

Output: Function returns the requested value on success, null if not valued.

5.2.6.2 Get Address

Routine: GETAD^HLOPRS2(.SEG,.VALUE,FIELD,COMP,REP)

Description: Gets an AD data type (Address) from the segment at the specified location. It can also be used to get the 1st 8 components of the XAD (Extended Address) data type.

Note: The XAD (extended address) data type replaced the AD data type as of version 2.3 of the HL7 standard. The XAD data type has additional components not included in the AD data type. If the additional components of the XAD data type are needed, GETAD^HLOPRS can be called, then the additional parts can be obtained by calling GET^HLOPRS.

Input:

SEG	Required	Pass-by-Reference	The parsed segment.
FIELD	Required	Pass-by-Value	The sequence # of the field.
COMP	Optional	Pass-by-Value	If specified, the address is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

VALUE	Required	Pass-by-Reference	These subscripts are returned: • "STREET1" -street address • "STREET2" - other designation • "CITY" • "STATE" - state or province • "ZIP" - zip or postal code • "COUNTRY" • "TYPE" - address type • "OTHER" - other geographic designation

5.2.6.3 Get Coded Element

Routine: GETCE^HLOPRS2(.SEG,.VALUE,FIELD,COMP,REP)

Description: Gets a CE data type (Coded Element) from the parsed segment at the specified location.

Input:

SEG	Required	Pass-by-Reference	The parsed segment.
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the coded element is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

VALUE	Required	Pass-by-Reference	These subscripts are returned: • "ID" - the identifier • "TEXT" - • "SYSTEM" - name of the code system • "ALTERNATE ID" - alternate identifier • "ALTERNATE TEXT" • ALTERNATE SYSTEM" - name of the alternate coding system

5.2.6.4 Get Coded Element with No Exceptions

Routine: GETCNE HLOPRS2(.SEG,.VALUE,FIELD,COMP,REP)

Description: Gets a CNE data type (Coded Element with No Exceptions) from the parsed segment at the specified location.

Input:

SEG	Required	Pass-by-Reference	The parsed segment.
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the coded element is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

VALUE	Required	Pass-by-Reference	These subscripts are returned: • "ID" - the identifier • "TEXT" - • "SYSTEM" - name of the code system • "ALTERNATE ID" - alternate identifier • "ALTERNATE TEXT" • ALTERNATE SYSTEM" - name of the alternate coding system • "SYSTEM VERSION" - version ID of the coding system • "ALTERNATE SYSTEM VERSION" - version ID of the alternate codin system • "ORIGINAL TEXT"

5.2.6.5 Get Coded Element with Exceptions

Routine: GETCWE^HLOPRS2(.SEG,.VALUE,FIELD,COMP,REP)

Description: Gets an CWE data type (Coded Element with Exceptions) from the parsed segment at the specified location.

Input:

SEG	Required	Pass-by-Reference	The parsed segment..
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the coded element is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

VALUE	Required	Pass-by-Reference	These subscripts are returned: • "ID" - the identifier • "TEXT" - • "SYSTEM" - name of the code system • "ALTERNATE ID" - alternate identifier • "ALTERNATE TEXT" • ALTERNATE SYSTEM" - name of the alternate coding system • "SYSTEM VERSION" - version ID of the coding system • "ALTERNATE SYSTEM VERSION" - version ID of the alternate codin system • "ORIGINAL TEXT"

5.2.6.6 Get Date

Routine: GETDT^HLOPRS2(.SEG,.VALUE,FIELD,COMP,REP)

Description: Gets a date from the parsed array at the specified location and converts it to FileMan format. The degree of precision is optionally returned.

Input:

SEG	Required	Pass-by-Reference	The parsed segment.
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the date is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

VALUE	Required	Pass-by-Reference Pass-by-Reference if the precision is needed	The date/time in FileMan format. The "PRECISION" subscript is also returned:
"PRECISION" – VALUE must be passed-by-reference if this subscript is needed Expected values are: • "S" – second (not valid for DT) • "M" – minute (not valid for DT) • "H" – hour (not valid for DT) • "D" - day • "L" - month • "Y" - year • "" - precision not specified			

5.2.6.7 Get Hierarchic Designator

Routine: GETHD^HLOPRS2(.SEG,.VALUE,FIELD,COMP,REP)

Description: Gets an HD data type (Hierarchic Designator) from the parsed segment at the specified location.

Input:

SEG	Required	Pass-by-Reference	The parsed segment..
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the hierarchic designator is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number. For non-repeating fields, this parameter is not necessary.

Output:

VALUE	Required	Pass-by-Reference	These subscripts are returned: • "NAMESPACE ID" • "UNIVERSAL ID" • "UNIVERSAL ID TYPE"

5.2.6.8 Get Timestamp

Routine: GETTS^HLOPRS2(.SEG,.VALUE,FIELD,COMP,REP)

Description: Gets a timestamp in HL7 format from the segment at the specified location and converts it to FileMan format. If the data type value includes the time zone, then the time is converted to local time. The degree of precision is optionally returned.

Input:

SEG	Required	Pass-by-Reference	The parsed segment.
FIELD	Required	Pass-by-Value	The sequence # of the field.
COMP	Optional	Pass-by-Value	If specified, the time stamp is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence #, defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

VALUE	Required	Pass-by-Reference IF subscripts are used	The date/time in FileMan format. The PRECISION subscript is optional, if provided the time stamp's precision will be determined.
"PRECISION" - Value should be Passed-by-Reference if the precision is needed. Expected values are: • "S" - second • "M" - minute • "H" - hour • "D" - day • "L" - month • "Y" - year • "" - precision not specified **Note:** FM does not allow greater precision than seconds, so greater precision will be rounded down to the second.			

5.2.6.9 Get Extended Person Name

Routine: GETXPN^HLOPRS2(.SEG,.VALUE,FIELD,COMP,REP)

Description: Gets an XPN data type (Extended Person Name) from the parsed array at the specified location.

Note: The XPN data type has additional components not included in this API. If needed, the additional components can be obtained by calling GET^HLOPRS after calling GETXPN^HLOPRS2.

This API can also be used to get the PN (Person Name) data type. The PN data type was made obsolete in version 2.5 of the HL7 standard.

Input:

SEG	Required	Pass-by-Reference	The parsed segment
FIELD	Required	Pass-by-Value	The sequence number of the field.
COMP	Optional	Pass-by-Value	If specified, the extended person name is presumed to be a component value with the parts as subcomponents. If not specified, the parts are presumed to be components of the field.
REP	Optional	Pass-by-Value	The occurrence number, which defaults to 1. For non-repeating fields, this parameter is not necessary.

Output:

VALUE	Required	Pass-by-Reference	These subscripts are returned: • "FAMILY" • "GIVEN" first name • "SECOND" second and further names or initials • "SUFFIX" (e.g., JR) • "PREFIX" (e.g., DR) • "DEGREE" (e.g., MD)

5.3 Returning Application Acknowledgments

5.3.1 Begin Application Acknowledgment for an Individual Message

Routine: $$ACK^HLOAPI2(.MSG,.PARMS,.ACK,.ERROR)

Description: Starts the process of building an application acknowledgement to return in response to an Individual mesage. This API should NOT be called for batch messages; instead use $$BATCHACK^HLOAPI3.

Input:

MSG		Required	Pass-by-Reference	Contains the original message. Its obtained by calling $$STARTMSG^HLOPRS.
PARMS Allowed subscripts:		Required	Pass-by-Reference	Contains various parameters that may be set to configure the application acknowledgment message.
	"ACK CODE"	Required	The value to return in MSA-1. Possible values are “AA”, “AE”, or “AR”.	
	"ERROR MESSAGE"	Optional	An optional error message to return in MSA-3, should be used only if the return code is AE or AR.	
	"ACCEPT ACK RESPONSE"	Optional	The <tag^routine> to call in response to a commit acknowledgment.	
	"ACCEPT ACK TYPE"	Optional	Indicates whether or not to request a commit acknowledgment. Possible values are “AL” or “NE”. Defaults to “AL”.	
	"CONTINUATION POINTER"	Optional	Indicates a fragmented message.	
	"COUNTRY"	Optional	The three-character country code	
	"EVENT"	Optional	The three-character event type which defaults to the event code of the original message.	

	"ENCODING CHARACTERS"	Optional	The four HL7 encoding characters which defaults to "^~\&".
	"FAILURE RESPONSE"	Optional	The <tag>^<routine> that the sending application routine should execute if the transmission of the message fails.
	"FIELD SEPARATOR"	Optional	Field separator which defaults to "|".
	"MESSAGE TYPE"	Optional	If not defined, ACK is used.
	"MESSAGE STRUCTURE CODE"	Optional	An optional code for the MSH-9 field.
	"QUEUE"	Optional	An application can name its own private queue (a string under 20 characters, it should be namespaced). The default is the name of the queue of the original message
	"SECURITY"	Optional	Security information to include in the header segment, SEQ-8.
	"VERSION"	Optional	The HL7 Version ID which defaults to 2.4.

Output: Function call returns 1 on success, 0 on failure.

PARMS			Killed when the function returns.
ACK	Required	Pass-by-Reference	The workspace where the acknowledgment message is being built.
ERROR	Optional	Pass-by-Reference	On encountering an error, returns a message.

5.3.2 Begin Batch Application Acknowledgement

Routine: $$BATCHACK^HLOAPI3(.MSG,.PARMS,.ACK,.ERROR)

Description: Used to initiate a batch application acknowledgement. It will contain individual application acknowledgements for each message in the original batch message that was received via HLO. Individual acknowledgements are placed in this batch by calling $$ADDACK^HLOAPI3, then the batch of acknowledgements is actually sent by calling $$SENDACK^HLOAPI2

Input:

MSG		Required	Pass-by-Reference	Obtained by calling $$STARTMSG^HLOPRS when parsing the original message. The application MUST NOT directly modify any values in this array.
PARMS Allowed subscripts:		Optional	Pass-by-Reference	Used to pass in various parameters.
	"ACCEPT ACK RESPONSE"	Optional		<tag^routine> to call in response to a commit ack.
		Optional		<AL,NE> which defaults to AL.

	"ACCEPT ACK TYPE"		
	"COUNTRY"	Optional	A three-character country code from the HL7 standard table.
	"ENCODING CHARACTERS"	Optional	The four HL7 encoding characters; optional, defaults to "^~\&".
	"FAILURE RESPONSE"	Optional	The <tag>^<routine> that the sending application routine should execute if the transmission of the message fails, i.e., the message cannot be sent or a requested commit ack is not received.
	"FIELD SEPARATOR"	Optional	Field separator; optional, defaults to "|".
	"QUEUE"	Optional	An application can name a private queue (a string under 20 characters, it should be namespaced). The default is the name of the queue of the original message.
	"SECURITY"	Optional	Security information to include in the header segment, SEQ-8.
	"VERSION"	Optional	The HL7 Version ID which defaults to 2.4.

Output: Function call - returns 1 on success, 0 on failure.

PARMS			Killed when the function returns.
ACK	Required	Pass-by-Reference	The acknowledgement message being built.
ERROR	Optional	Pass-by-Reference	An error message.

5.3.3 Add an Application Acknowledgement to a Batch

Routine: $$ADDACK^HLOAPI3(.ACK,.PARMS,.ERROR)

Description: Used to add an application acknowledgement to a batch acknowledgement message that was started by calling $$BATCHACK^HLOAPI3.

Input:

ACK		Required	Pass-by-Reference	The batch of acknowledgements that is being built.
PARMS Allowed subscripts:		Required	Pass-by-Reference	Used to pass in various parameters.
	"ACK CODE"	Required	Allowed values are AA, AE, or AR, and is used to populate MSA-1.
	"ERROR MESSAGE"	Optional	If the "ACK CODE" is AE or AR, then the value passed in this parameter is used to populate MSA-3.
	"EVENT"	Optional	A three-character event type (optional, defaults to the event type of the original message).
	"MESSAGE CONTROL ID"	Required	The message control ID of the original individual message within the batch that is being acknowledged.

	"MESSAGE STRUCTURE CODE"	Optional	
	"MESSAGE TYPE"	Optional	A three-character message type that defaults to ACK.
	"SECURITY"	Optional	Security information to include in the header segment SEQ-8.

Output: (Function call returns 1 on success, 0 on failure.)

PARMS			Killed when the function returns.
ACK	Required	Pass-by-Reference	The acknowledgement message being built.
ERROR	Optional	Pass-by-Reference	An error message.

5.3.4 Send the Application Acknowledgement.

Routine: $$SENDACK^HLOAPI2(.ACK,.ERROR)

Description: Used to send the acknowledgement message that was initiated by a call to $$ACK^HLAPI2 or a batch of acknowledgement messages that was initiated by a call to $$BATCHACK^HLOAPI3.

Input:

ACK	Required	Pass-by-Reference	An array that contains the acknowledgement message.

Output: Function Call - Returns 1 on success, 0 on failure.

ERROR	Optional	Pass-by-Reference	An error message is returned if the function fails.

5.4 Subscription Registry

An entry in the HLO Subscription Registry is similar to a mailing list in that it can be used to send HL7 messages to multiple recipients. The application that creates the entry is the owner and is responsible for maintaining it.

5.4.1 Create an Entry in the Subscription Registry

Routine: $$CREATE^HLOASUB(OWNER,DESCRIPTION,.ERROR)

Description: Used to create a new entry in the HLO SUBSCRIPTION REGISTRY File (#779.4).

Input:

OWNER	Required	Pass-by-Value	The name of the owning application. It should be prefixed with the package namespace to ensure uniqueness.
DESCRIPTION	Required	Pass-by-Value	A brief (1 line) description.

Output: Function call returns the new IEN in the HLO SUBSCRIPTION REGISTRY File (#779.4) if successful, 0 if error.

ERROR	Optional	Pass-by-Referen	An error message is returned if the function fails.

5.4.2 Add a New Recipient to a Subscription Registry Entry

Routine: $$ADD^HLOASUB(IEN,.WHO,.ERROR)

Description: Used to add a new recipient to the subscription list.

Input:

IEN	Required	Pass-by-Value	The IEN of the entry in the HLO SUBSCRIPTION REGISTRY File (#779.4).
WHO	Required	Pass-by-Reference	Used to define a new subscriber.
These subscripts are allowed:			
"RECEIVING APPLICATION"	Required	String, 60 char max, required.	
	ONE of the following four parameters **MUST** be provided to identify the Receiving Facility:		
"FACILITY LINK IEN"	Optional	IEN of the logical link.	
"FACILITY LINK NAME"	Optional	Name of the logical link.	
"INSTITUTION IEN"	Optional	Pointer to the INSTITUTION File (#4).	
"STATION NUMBER"	Optional	Station number with suffix.	
	ONE of the following two parameters **MAY** be provided - optionally - to identify the middleware to route the message through, such as an interface engine:		
“MIDDLEWARE LINK IEN")	Optional	Pointer to a logical link for that is setup to communicate with the middleware.	
“MIDDLEWARE LIN NAME"	Optional	Name of the link to communicate with the middleware.	

Output: Function call returns the IEN of the recipient from the RECIPIENTS multiple, 0 on failure.

WHO			Killed when the function returns.
ERROR	Optional	Pass-by-Reference	On an error, one of the following error messages may be returned: • "SUBSCRIPITON REGISTRY ENTRY NOT FOUND" • "RECEIVING FACILTY LOGICAL LINK NOT FOUND" • "RECEIVING APPLICATION NOT FOUND" • "MIDDLEWARE LOGICAL LINK PROVIDED BUT NOT FOUND" • "FAILED TO ACTIVATE SUBSCRIBER"

5.4.3 Terminate a Recipient from a Subscription Registry Entry

Routine: $$END^HLOASUB(IEN,.WHO)

Description: Used to terminate a recipient from the subscriber list. The recipient is not deleted, but the DATE/TIME TERMINATED field is entered with the current date/time.

Input:

IEN	Required	Pass-by-Value	The IEN of the HLO SUBSCRIPTION REGISTRY File (#779.4) entry.
WHO	Required	Pass-by-Reference	If WHO("SUBIEN") is defined, then it should be the IEN of the sub-record to be terminated. Otherwise, set the parameters as per $$ADD^HLOASUB.

Output: Function call returns 1 on success, 0 on failure.

WHO			Killed when the function returns.

5.4.4 Check Subscription Registry Entry for a Recipient

Routine: $$ONLIST^HLOASUB(IEN,LINKIEN,APPNAME,FAC1,FAC2,FAC3)

Description: Used to locate an individual recipient within a subscription registry entry.

Input:

IEN	Required	Pass-by-Value	The IEN of the HLO SUBSCRIPTION REGISTRY File (#779.4) entr
LINKIEN	Required	Pass-by-Value	IEN of the logical link.
APPNAME	Required	Pass-by-Value	Name of the receiving application.
FAC1	Required	Pass-by-Value	Component 1 of the receiving facility.
FAC2	Required	Pass-by-Value	Component 2 of the receiving facility.
FAC3	Required	Pass-by-Value	Component 3 of the receiving facility.

Output: Function call returns the IEN of the recipient from the RECIPIENTS multiple, 0 on failure.

5.4.5 Retrieve Next Recipient from a Subscription Registry Entry

Routine: $$NEXT^HLOASUB(IEN,.RECIP)

Description: Used to loop through a subscription list. It ignores recipients that have been terminated from the specified subscription registry entry.

Input:

IEN	Required	Pass-by-Value	The IEN of the HLO SUBSCRIPTION REGISTRY File (#779.4) entry.
RECIP	Required	Pass-by-Reference	If empty, the function finds the first recipient in the specified subscription registry entry, else it uses the value of RECIP("SUBIEN") to find the next recipient.

Output: Function call - Returns the IEN of the recipient from the RECIPIENTS multiple, 0 on Failure. If no more recipients are found, then SUBIEN=-1 and all other subscripts are set to null.

RECIP These subscripts are returne	Required	Pass-by-Reference	Returns the next recipient in the specified subscription registry entry.
	“LINK IEN"	IEN of the logical link	
	"LINK NAME"	Name of the logical link	
	"RECEIVING APPLICATION”	Name of receiving application	
	("RECEIVING FACILITY",1)	Component 1 of receiving facility	
	("RECEIVING FACILITY",2)	Component 2 of receiving facility	
	("RECEIVING FACILITY",3)	Component 3 of receiving facility	

▨	"SUBIEN"	The IEN in the multiple, used to find the next recipient in the specified subscription registry entry

5.4.6 Build a Subscription Registry Index

Routine: $$INDEX^HLOASUB1(IEN,.PARMS)

Description: Used to build an index of its subscriptions. This is optional, but using this function allows the application to find subscriptions without storing the IEN.

Input:

IEN	Required	Pass-by-Value	The IEN of the HLO SUBSCRIPTION REGISTRY File (#779.4) entry.
PARMS	Required	Pass-by-Reference	An array of parameters with which to build the index. The format is: PARMS(1)=<first parameter>, PARMS(2)=<second parameter> If PARMS(i)=null, the parameter is translated to a single space.

Output: Function Call - Returns 1 on success, 0 on failure.

PARMS			Killed when the function returns.

5.4.7 Find a Subscription Registry Entry

Routine: $$FIND^HLOASUB1(OWNER,.PARMS)

Description: Used to find a subscription registry entry. The application must maintain a private index via $$INDEX^HLOASUB1 in order to utilize this function,

Input:

OWNER	Required	Pass-by-Value	The name of the owning application, as entered in $$CREATE^HLOSUB.
PARMS	Required	Pass-by-Reference	An array of parameters with which to build the index. The forn is: PARMS(1)=<first parameter>, PARMS(2)=<second parameter> If PARMS(i)=null, the parameter is translated to a single space

Output: Function returns the IEN of the subscription list if found, 0 otherwise.

PARMS			Killed when the function returns.

5.5 Miscellaneous APIs

5.5.1 Resend a Message

Routine: $$RESEND^HLOAPI3(MSGIEN,.ERROR)

Description: Used to retransmit a message by making a copy of the message, reusing all the original parameters. Then the message is placed in the same outgoing queue as the original message.

Input:

MSGIEN	Required	Pass-by-Va	The IEN of the HLO MESSAGES file (#778) entry that is to be resent.

Output: Function returns the IEN of the new message in HLO MESSAGES file (#778) on success, 0 on failure.

ERROR	Optional	Pass–by-Referei	On failure, will return an error message.

5.5.2 Reprocess a Message

Routine: $$REPROC^HLOAPI3(MSGIEN,.ERROR)

Description: Used to place a message back on an incoming queue for reprocessing by the application. The HLO in-filer process will execute the applications routine in the background.

Input:

MSGIEN	Required	Pass-by-Value	The IEN of the HLO MESSAGES file (#778) entry that is to be reprocessed.

Output: Function Call - Rreturns 1 on success, 0 on failure

ERROR	Optional	Pass-by-Referei	On failure, will contain an error message.

5.5.3 Reprocess a Message Immediately

Routine: $$PROCNOW^HLOAPI3(MSGIEN,.ERROR)

Description: Used to pass a message immediately to the application for processing. It differs from $$REPROC^HLOAPI3 in that it does not put the message on the incoming queue for the in-filer to process, instead it executes the application routine immediately.

Input:

MSGIEN	Required	Pass-by-Value	The IEN of the HLO MESSAGES file (#778) entry that is to be reprocessed.

Output: Function Call - Returns 1 on success, 0 on failure.

ERROR	Optional	Pass-by-Reference	On failure, will contain an error message.

5.5.4 Reset the Purge Date and Time for a Message

Routine: $$SETPURGE^HLOAPI3(MSGIEN,TIME)

Description: Used to reset the scheduled purge date/time of a message.

Input:

MSGIEN	Required	Pass-by/Value	The IEN of the HLO MESSAGES file (#778) entry that is to be modified.
TIME	Optional	Pass-by-Reference	The new scheduled purge date/time. If not defined, defaults to NOW.

Output: Function returns 1 on success, 0 on failure.

5.5.5 Count Messages on All Queues

Routine: $$QUECNT^HLOQUE(.ARRAY)

Description: Returns the count of messages pending on all incoming, outgoing, and sequence queues.

Input: None

Output:

ARRAY	Required	Pass-by-Reference	The array will return a list of all queues and the message count on each. The format is: ARRAY("TOTAL") = Total number of messages on all queues. ARRAY("OUT") = Total number of outgoing messages ARRAY("IN") = Total number of incoming messages. ARRAY("SEQ")= Total number of messages on sequence queues. ARRAY("IN",<link_name>,<queue_name>) = Number of messages on given link and queue. ARRAY("OUT",link_name,queue_name) = Number of messages on given link and queue. ARRAY("SEQ",<queue_name>) = Number of messages on the given sequence queue.

5.5.6 Count Messages on Outgoing Queues

Routine: $$OUT^HLOQUE(.ARRAY)

Description: Returns the count of messages pending on all outgoing queues.

Input: None.

Output:

ARRAY	Required	Pass-by-Reference	ARRAY will return the list of outbound queues and the message count on each. The format is: ARRAY("OUT") = Total number of outgoing messages. ARRAY("OUT",<link_name>,<queue_name>) = Number of messages on the given link and queue.

5.5.7 Count of Messages on Incoming Queues

Routine : $$IN^HLOQUE(.ARRAY)

Description: Returns the count of messages pending on all incomming queues.

Input: None

Output:

ARRAY	Required	Pass-by-Reference	ARRAY will return a list of the inbound queues and the message count on each. The format is: ARRAY("IN") = Total number of outgoing messages. ARRAY("IN",<sending facility>,<queue_name>) = Number of messages on the given link and queue.

5.5.8 Count of the number of messages on the sequence queues.

Routine: $$SEQ^HLOQUE(.ARRAY)

Description : Returns the total number of messages pending on the incoming queues.

Input: None.

Output:

ARRAY	Required	Pass-by-Reference	Array will return a list of all the sequences queues. The format is: ARRAY("SEQ") = Total number of sequence queue messages. ARRAY("SEQ",<link_name>,<queue_name>) = Number of messages on given link and queue.

6.0 Code Examples

6.1 HLODEM1

```
HLODEM1 ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**146**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;
PID(DFN,SEQ,SEG) --
    ;
    ;Description:
    ; Builds the PID segment using the HLO segment building APIs.
    ; PIMS APIs are called to obtain data from PATIENT file (#2).
    ;
    ; The fields that are included in the segment are:
    ;  PID-1 Set ID
    ;  PID-3 Patient Identifier List:  This is a repeating field.
    ;   The first repetition will be set to the ICN, the second to
    ;   the SSN.
    ;  PID-5 Patient Name
    ;  PID-7 Date/Time of Birth
    ;  PID-11 Patient Address
    ;
    ;Input:
    ; DFN (required) The IEN of the record in the PATIENT file (#2).
    ; SEQ (optional) Value for PID-1, the Set ID field. For the first
    ;   occurrence of the PID segment it should be set to 1, for the
    ;   second 2, etc.
    ;
    ;Output:
    ;  SEG (pass-by-reference) The segment, returned as a list of fields.
    ;
    ;
    N VA,VADM,VAHOW,VAROOT,VATEST,VAPA,NAME,DOB,SSN,ICN,ADDRESS
    K SEG S SEG="" ;The segment should start off blank.
    ;
    ;Get the patient data using PIMS utilities.
    ;
    S VAHOW=1
    D DEM^VADPT
    S NAME=VADM("NM") ;The name returned in non-standard (VHA) format.
    ;
    ;Standardize the format of the name using a Kernel utility.
    ;Returns the subscripts "FAMILY","GIVEN","MIDDLE","SUBSCRIPT".
    D STDNAME^XLFNAME(.NAME,"C")
    ;
    S DOB=$P(VADM("DB"),"^") ;in FileMan format
    S SSN=$P(VADM("SS"),"^")
    ;
    ;Get the address.
    S VAHOW=""
    D ADD^VADPT
    ;Move address components into ADDRESS.
    S ADDRESS("STREET1")=VAPA(1)
```

```
 S ADDRESS("STREET2")=VAPA(2)
 S ADDRESS("CITY")=VAPA(4)
 S ADDRESS("STATE")=$P(VAPA(5),"^",2)
 S ADDRESS("ZIP")=VAPA(6)
 ;
 ;Call an MPI utility to get the ICN.
 S ICN=$P($$GETICN^MPIF001(DFN),"V")
 ;
 ;Use the HLO APIs to set the data into the segment.
 ;
 D SET^HLOAPI(.SEG,"PID",0) ;Set the segment type.
 D SET^HLOAPI(.SEG,SEQ,1) ;Set PID-1.
 ;
 ;Set ICN into PID-3, repetition 1.
 D SET^HLOAPI(.SEG,ICN,3,1,1,1) ;component 1, subcomponent 1
 D SET^HLOAPI(.SEG,"USVHA",3,4,1,1) ;component 4, subcomponent 1
 D SET^HLOAPI(.SEG,"0363",3,4,3,1) ;component 4, subcomponent 3
 D SET^HLOAPI(.SEG,"NI",3,5,1,1) ;component 5
 ;
 ;Set SSN into PID-3, repetition 2.
 D SET^HLOAPI(.SEG,SSN,3,1,1,2) ;component 1, subcomponent 1
 D SET^HLOAPI(.SEG,"USSSA",3,4,1,2) ;component 4, subcomponent 1
 D SET^HLOAPI(.SEG,"0363",3,4,3,2) ;component 4, subcomponent 3
 D SET^HLOAPI(.SEG,"SS",3,5,1,2) ;component 5
 ;
 ;Set the name into PID-5.
 D SETXPN^HLOAPI4(.SEG,.NAME,5)
 ;
 ;Set DOB into PID-7.
 D SETDT^HLOAPI4(.SEG,DOB,7)
 ;
 ;Set the address into PID-11.
 D SETAD^HLOAPI4(.SEG,.ADDRESS,11)
 Q
 ;
NK1(DFN,SEQ,SEG) --
 ;
 ;Description:
 ; Builds the NK1 segment for the primary emergency contact
 ; using the HLO segment building APIs. A PIMS API is called to get the
 ; necesary data from PATIENT file (#2).
 ;
 ; The fields included in the segment are:
 ;  NK1-1 Set ID: Set to the SEQ input parameter. For the 1st
 ;          occurrence of the NK1 segment it should be set
 ;          to 1, for the 2nd 2, etc.
 ;  NK1-2 Name
 ;  NK1-3 Relationship
 ;  NK1-4 Address
 ;  NK1-5 Phone Number
 ;  NK1-7 Contact Role
 ;
 ;Input:
 ; DFN (required) The IEN of the record in the PATIENT file (#2).
 ; SEQ (optional) Value for NK1-1.
 ;
 ;Output:
    ;  SEG (pass-by-reference) Will return an array containing the segment.
    ;    The ADDSEG^HLOAPI API must be called to move the segment into
    ;    the message.
    ;
    N VA,VAOA,VAHOW,VAROOT,VATEST,NAME,ADDRESS
    K SEG S SEG="" ;The segment should start off blank.
    ;
    ;Get the patient's emergency contact using a PIMS utility.
    S VAOA("A")=1
    D OAD^VADPT
    ;
    S NAME=VAOA(9) ;The name is returned in non-standard (VHA) format.
    ;
    ;Standardize the format of the name using a Kernel utility.
    ;Returns the subscripts "FAMILY","GIVEN","MIDDLE","SUBSCRIPT".
    D STDNAME^XLFNAME(.NAME,"C")
    ;
    ;Move the address components into ADDRESS.
    S ADDRESS("STREET1")=VAOA(1)
    S ADDRESS("STREET2")=VAOA(2)
    S ADDRESS("CITY")=VAOA(4)
    S ADDRESS("STATE")=$P(VAOA(5),"^",2)
    S ADDRESS("ZIP")=VAOA(6)
    ;
    ;Now set the data into the segment.
    ;
    D SET^HLOAPI(.SEG,"NK1",0) ;Set the segment type.
    D SET^HLOAPI(.SEG,SEQ,1) ;Set NK1-1.
    ;
    ;Set the name into NK1-2.
    D SETXPN^HLOAPI4(.SEG,.NAME,2)
    ;
    ;Set the relationship into NK1-3, component 2.
    D SET^HLOAPI(.SEG,VAOA(10),3,2)
    ;
    ;Set the address into NK1-4.
    D SETAD^HLOAPI4(.SEG,.ADDRESS,4)
    ;
    ;Set the phone number into NK1-5.
    D SET^HLOAPI(.SEG,VAOA(8),5)
    ;
    ;Set the contact role into NK1-7.
    D SET^HLOAPI(.SEG,"EP",7,1)
    D SET^HLOAPI(.SEG,"EMERGENCY CONTACT PERSON",7,2)
    D SET^HLOAPI(.SEG,"0131",7,3)
    Q
A08(DFN,ERROR) --
    ;
    ;Description:
    ; Builds an ADT~A08 message and queues it for transmission.
    ; Included segments are the PID and NK1.
    ;
    ;Input:
    ;  DFN (required) ien of a patient record in the PATIENT file (#2)
    ;Output:
    ;  function:
    ;   On Success: Returns the ien of the messgage in the
```

```
     ;         HLO MESSAGES file (#778).
     ;   On Failure: Returns 0.
     ;  ERROR - (optional, pass-by-refernce) On failure returns an
     ;      error message.
     ;
     ;Required Setup:
     ;<<HLO APPLICATION PARAMETER - file #779.2>>
     ; ** The sending application. **
     ; APPLICATION NAME: HLO DEMO SENDING APPLICATION
     ; ** The package that is sending the message. **
     ; Package File Link: HL7 OPTIMIZED (HLO)
     ;
     ;<<HL Logical Link, file #870>>
     ;  ** The receiving facility. **
     ;NODE: HLODEMO
     ; INSTITUTION: <Institution entered here.>
     ; LLP TYPE: TCP
     ; MAILMAN DOMAIN: <The use of DNS DOMAIN is preferred for new links.>
     ; DNS DOMAIN: <TCP/IP domain name for DNS entered here.>
     ; TCP/IP ADDRESS: <IP address entered here.>
     ; TCP/IP SERVICE TYPE: CLIENT (SENDER)
     ; TCP/IP PORT (OPTIMIZED): <Port # entered here.>
     ;
     ;
     N SEG,PARMS,WHOTO
     S PARMS("MESSAGE TYPE")="ADT"
     S PARMS("EVENT")="AO8"
     I '$$NEWMSG^HLOAPI(.PARMS,.MSG,.ERROR) Q 0
     D PID(DFN,1,.SEG)
     I '$$ADDSEG^HLOAPI(.MSG,.SEG,.ERROR) Q 0
     D NK1(DFN,1,.SEG)
     I '$$ADDSEG^HLOAPI(.MSG,.SEG,.ERROR) Q 0
     S PARMS("SENDING APPLICATION")="HLO DEMO SENDING APPLICATION"
     S WHOTO("RECEIVING APPLICATION")="HLO DEMO RECEIVING APPLICATION"
     S WHOTO("FACILITY LINK NAME")="HLODEMO"
     Q $$SENDONE^HLOAPI1(.MSG,.PARMS,.WHOTO,.ERROR)
     ;
BATCHA08(DFNLIST,ERROR) --
     ;
     ;Description:
     ; Builds a batch of ADT~A08 messages and queues it for transmission.
     ; The DFNLIST is a list of patients to include messages for.
     ;
     ;Input:
     ; DFNLIST (required, pass-by-reference) A list of patient DFNs in the
     ;      format DFNLIST(<DFN)="".
     ;Output:
     ; Function:
     ;   On Success: Returns the ien of the messgage in the
     ;         HLO MESSAGES file (#778).
     ;   On Failure: Returns 0.
     ;  ERROR (optional, pass-by-refernce) On failure returns an error
     ;     message.
     ;
     ;Required Setup: Same as for A08^HLODEM1
     ;
     N MSG,PARMS,SEG,WHOTO,MSG,DFN,QUIT
```

```
    I '$$NEWBATCH^HLOAPI(,.MSG,.ERROR) Q 0
    S (DFN,QUIT)=0
    F S DFN=$O(DFNLIST(DFN)) Q:(QUIT!('DFN)) D
    .N PARMS
    .S PARMS("MESSAGE TYPE")="ADT"
    .S PARMS("EVENT")="AO8"
    .I '$$ADDMSG^HLOAPI(.MSG,.PARMS,.ERROR) S QUIT=1 Q
    .D PID(DFN,1,.SEG)
    .I '$$ADDSEG^HLOAPI(.MSG,.SEG,.ERROR) Q 0
    .D NK1(DFN,1,.SEG)
    .I '$$ADDSEG^HLOAPI(.MSG,.SEG,.ERROR) Q 0
    ;
    S PARMS("SENDING APPLICATION")="HLO DEMO SENDING APPLICATION"
    S WHOTO("RECEIVING APPLICATION")="HLO DEMO RECEIVING APPLICATION"
    S WHOTO("FACILITY LINK NAME")="HLODEMO"
    Q $$SENDONE^HLOAPI1(.MSG,.PARMS,.WHOTO,.ERROR)
    Q
```

6.2 HLODEM2

```
HLODEM2 ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**146**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;
PID(DFN,SEQ,SEG) --
    ;
    ;Description: Builds the PID segment exactly as in PID^HLODEM1, but
    ; without using the HLO APIs for setting the values into the segment.
    ;
    ;Input:
    ; DFN (required) The IEN of the record in the PATIENT file (#2).
    ; SEQ (optional) Value for PID-1, Set ID. For the first occurence
    ;   of the segement it should be set to 1, for the second 2, etc.
    ;
    ;Output:
    ;  SEG (pass-by-reference) The PID segment as a single line.
    ;
    ;Notes: 1) Assumes that the variables returned from INIT^HLFNC2 are
    ;     defined, specifically, HL("FS") and HL("ECH").
    ;    2) This segment builder takes the shortcut of not replacing
    ;     delimiters that occur within data fields with their
    ;     escape sequences.
    ;    3) Also takes the shortcut of assuming the segment will fit
    ;     on a single node.
    ;
    N VA,VADM,VAHOW,VAROOT,VATEST,VAPA,NAME,DOB,SSN,ICN,ADDRESS,FS,CS,RS,SS
    K SEG S SEG="" ;The segment should start off blank.
    ;
    ;Set the message delimiters for easy reference.
    S FS=HL("FS") ;field separator
    S CS=$E(HL("ECH"),1) ;component separator
    S RS=$E(HL("ECH"),2) ;repitition separator
    S SS=$E(HL("ECH"),4) ;subcomponent separator
    ;
    ;Get the patient data using a PIMS utility.
    S VAHOW=1
    D DEM^VADPT
    S NAME=VADM("NM") ;The name is returned in non-standard (VHA) format.
    ;
    ;Standardize the format of the name using a Kernel utility.
    ;Returns the subscripts "FAMILY","GIVEN","MIDDLE","SUBSCRIPT".
    D STDNAME^XLFNAME(.NAME,"C")
    ;
    S DOB=$P(VADM("DB"),"^") ;(in FileMan format)
    S SSN=$P(VADM("SS"),"^")
    ;
    ;Get the address.
    S VAHOW=""
    D ADD^VADPT
    ;Move address components into ADDRESS.
    S ADDRESS("STREET1")=VAPA(1)
    S ADDRESS("STREET2")=VAPA(2)
    S ADDRESS("CITY")=VAPA(4)
    S ADDRESS("STATE")=$P(VAPA(5),"^",2)
    S ADDRESS("ZIP")=VAPA(6)
    ;
    ;Call an MPI utility to get the ICN.
    S ICN=$P($$GETICN^MPIF001(DFN),"V")
    ;
    ;Now concatenate all the fields together that make up the PID segment.
    S SEG="PID"_FS_SEQ_FS_FS_ICN_CS_CS_CS_"USVHA"_SS_SS_"0363"_CS_"NI"
    S SEG=SEG_RS_SSN_CS_CS_CS_"USSSA"_SS_SS_"0363"_CS_"SS"
    S SEG=SEG_FS_FS_NAME("FAMILY")_CS_NAME("GIVEN")_CS_NAME("MIDDLE")_CS_NAM
    E("SUFFIX")
    ;
    ;DOB needs to be converted to HL7 format.
    S DOB=$$HLDATE^HLFNC(DOB,"DT")
    ;
    S SEG=SEG_FS_FS_DOB_FS_FS_FS_FS_ADDRESS("STREET1")_CS_ADDRESS("STREET2")
    _CS_ADDRESS("CITY")_CS_ADDRESS("STATE")_CS_ADDRESS("ZIP")
    ;
    Q
    ;
NK1(DFN,SEQ,SEG) --
    ;
    ;Description: Builds the NK1 segment for the primary emergency
    ; contact as in NK1^HLODEM1, but without using the HLO segment building
    ; APIs. A PIMS API is used to get the necesary data from PATIENT
    ; file (#2).
    ;
    ; The fields included in the segment are:
    ;  NK1-1 Set ID: Set to the SEQ input parameter.
    ;  NK1-2 Name
    ;  NK1-3 Relationship
    ;  NK1-4 Address
    ;  NK1-5 Telephone Number
    ;  NK1-7 Contact Roll
    ;
    ;Input:
    ; DFN (required) The IEN of the record in the PATIENT file (#2).
    ; SEQ (optional) Value for NK1-1.
    ;
    ;Output:
    ;  SEG (pass-by-reference) The NK1 segment as a single line.
    ;
    ;Notes: 1) Assumes that the variables returned from INIT^HLFNC2 are
    ;     defined, specifically, HL("FS") and HL("ECH").
    ;    2) This segment builder takes the shortcut of not replacing
    ;     delimiters that occur within data fields with their
    ;     escape sequences.
    ;    3) Also takes the shortcut of assuming the segment will fit
    ;     on a single node.
    ;
    ;
    N VA,VAOA,VAHOW,VAROOT,VATEST,NAME,ADDRESS,FS,CS
    K SEG S SEG="" ;The segment should start off blank.
    ;
    ;Set the message delimiters for convenience.
    S FS=HL("FS") ;Field Separator
    S CS=$E(HL("ECH"),1) ;Component Separator
    ;
    ;Get the patient's emergency contact using a PIMS utility.
    S VAOA("A")=1
    D OAD^VADPT
    ;
    S NAME=VAOA(9) ;(name returned in non-standard (VHA) format)
    ;
    ;Standardize the format of the name using a Kernel utility.
    ;returns the subscripts "FAMILY","GIVEN","MIDDLE","SUBSCRIPT"
    D STDNAME^XLFNAME(.NAME,"C")
    ;
    ;Move address components into ADDRESS.
    S ADDRESS("STREET1")=VAOA(1)
    S ADDRESS("STREET2")=VAOA(2)
    S ADDRESS("CITY")=VAOA(4)
    S ADDRESS("STATE")=$P(VAOA(5),"^",2)
    S ADDRESS("ZIP")=VAOA(6)
    ;
    ;Set the data into the segment.
    ;
    ;Set segment type,set id, and name.
    S SEG="NK1"_FS_1_FS_NAME("FAMILY")_CS_NAME("GIVEN")_CS_NAME("MIDDLE")_CS
    _NAME("SUFFIX")
    ;
    ;Set the relationship into NK1-3, component 2.
    S SEG=SEG_FS_CS_VAOA(10)
    ;
    ;Set the address into NK1-4.
    S SEG=SEG_FS_ADDRESS("STREET1")_CS_ADDRESS("STREET2")_CS_ADDRESS("CITY")
    _CS_ADDRESS("STATE")_CS_ADDRESS("ZIP")
    ;
    ;Set the phone number into NK1-5.
    S SEG=SEG_FS_VAOA(8)
    ;
    ;Set the contact role into NK1-7.
    S SEG=SEG_FS_"EP"_CS_"EMERGENCY CONTACT PERSON"_CS_"0131"
    Q
```

```
     ;
A08(DFN,ERROR) --

     ;Description: Builds an ADT~A08 message and queues it for transmission.
     ;  The transmission is via HLO, but uses an event protocol setup and
     ;  hardcoded segment builders. Included segments are the PID and NK1.
     ;
     ;Input:
     ;  DFN (required) ien of a patient record in the PATIENT file (#2)
     ;Output:
     ;  Function Return Value:
     ;   On Success: Returns the ien of the messgage in the HLO Messages
     ;         file (#778).
     ;   On Failure: Returns 0.
     ;  ERROR (optional, pass-by-refernce) On failure, returns an error
     ;     message.
     ;
     ;Required Setup:
     ;<<Subscriber protocol (file #101)>>
     ; ** The receiving application. **
     ; NAME: HLO DEMO A08 CLIENT   TYPE: subscriber
     ; RECEIVING APPLICATION: HLO DEMO RECEIVING APPLICATION
     ; LOGICAL LINK: HLODEMO
     ;
     ;<<Event protocol (file #101)>>
     ; ** The sending application. **
     ; NAME: HLO DEMO A08 SERVER   TYPE: event driver
     ; CREATOR: MOORE,JIM      SENDING APPLICATION: HLO DEMO SENDING AP
     PLICATION
     ; TRANSACTION MESSAGE TYPE: ADT     EVENT TYPE: A08
     ; ACCEPT ACK CODE: AL     APPLICATION ACK TYPE: NE
     ; VERSION ID: 2.4
     ; SUBSCRIBERS: HLO DEMO A08 CLIENT
     ;
     ;<<HL7 Application Parameter (file #771) for receiving application>>
     ; NAME: HLO DEMO RECEIVING APPLICATION
     ; ACTIVE/INACTIVE: ACTIVE
     ;
     ;<<HL7 Application Parameter - file #771>>
     ; ** Represents the sending application. This will be automatically
     ;   created if not already by HLO .**
     ; NAME: HLO DEMO SENDING APPLICATION
     ; ACTIVE/INACTIVE: ACTIVE
     ;
     ;<<HL Logical Link, file 870>>
     ; ** The receiving facility. **
     ; NODE: HLODEMO
     ; INSTITUTION: <institution entered here>
     ; LLP TYPE: TCP
     ; MAILMAN DOMAIN: <use of DNS DOMAIN is preferred for new links>
     ; DNS DOMAIN: <TCP/IP domain name for DNS entered here>
     ; TCP/IP ADDRESS: <ip address entered here>
     ; TCP/IP SERVICE TYPE: CLIENT (SENDER)
     ; TCP/IP PORT (OPTIMIZED): <port # entered here>
     ;
     N SEG,HLA,HL,RESULT,RSLT
     D INIT^HLFNC2("HLO DEMO A08 SERVER",.HL)
```

```
    K ERROR
    D PID(DFN,1,.SEG)
    S HLA("HLS",1)=SEG
    D NK1(DFN,1,.SEG)
    S HLA("HLS",2)=SEG
    S RSLT=$$EN^HLOCNRT("HLO DEMO A08 SERVER","LM",,,.RESULT)
    I 'RSLT S ERROR=$P(RESULT,"^",4,5) Q 0
    Q RESULT("IEN")
```

6.3 HLODEM3

```
HLODEM3 ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**146**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;
A08(DFN,ERROR) --
    ;
    ;Description: Demonstrates the use of MOVESEG^HLOAPI to use hardcoded
    ; segment builders in conjunction with HLO. Builds an ADT~A08
    ; message and queues it for transmission. Included segments are the PID
    ; built with the HLO APIs and the NK1 segment built in the traditional
    ; hardcoded manner. The NK1 segment is moved into the HLO message by
    ; calling MOVESEG^HLOAPI. Protocols are not used.
    ;
    ;Input:
    ; DFN (required) The IEN of a patient record in the PATIENT file (#2).
    ;Output:
    ;  Function Return Value:
    ;   On Success: Returns the ien of the messgage in the HLO Messages
    ;         file (#778).
    ;   On Failure: Returns 0.
    ;  ERROR (optional, pass-by-reference) On failure returns an error
    ;     message.
    ;
    ;Required Setup: << Same as for A08^HLODEM1. >>
    ;
    ;
    N SEG,PARMS,WHOTO,HLA,HL
    S PARMS("MESSAGE TYPE")="ADT"
    S PARMS("EVENT")="A08"
    I '$$NEWMSG^HLOAPI(.PARMS,.MSG,.ERROR) Q 0
    ;
    ;Use the HLO segment building APIs to build the PID segment.
    D PID^HLODEM1(DFN,1,.SEG)
    ;
    ;Segments build with the HLO APIs are moved into the message by calling
    ;$$ADSEG^HLOAPI.
    I '$$ADDSEG^HLOAPI(.MSG,.SEG,.ERROR) Q 0
    ;
    ;Call the hardcoded NK1 segment builder.
    ;It requires that message delimiters be defined via HL("FS") and HL("ECH
    ").
    S HL("FS")="|"
    S HL("ECH")="^~\&"
    D NK1^HLODEM2(DFN,1,.SEG)
    ;
    ;Hardcoded segments are moved into the message by calling
    ;MOVESEG^HLOAPI.
    I '$$MOVESEG^HLOAPI(.MSG,.SEG,.ERROR) Q 0
    ;
    S PARMS("SENDING APPLICATION")="HLO DEMO SENDING APPLICATION"
    S WHOTO("RECEIVING APPLICATION")="HLO DEMO RECEIVING APPLICATION"
    S WHOTO("FACILITY LINK NAME")="HLODEMO"
    Q $$SENDONE^HLOAPI1(.MSG,.PARMS,.WHOTO,.ERROR)
    ;
A082(DFN,ERROR) --
    ;
    ;Description: Demonstrates the use of the MOVEMSG^HLOAPI to use
    ;traditional style message builders in conjunction with HLO. Builds an
    ;ADT~A08 message and queues it for transmission. The message includes
    ;the PID and NK1 segments and is built in the traditional manner as an
    ;array of hardcoded segments. The message is then moved to HLO by
    ;calling MOVEMSG^HLOAPI. Protocols are not used.
    ;
    ;Input:
    ; DFN (required) ien of a patient record in the PATIENT file (#2)
    ;Output:
    ;  Function Return Value:
    ;   On Success: Returns the IEN of the messgage in the HLO Messages
    ;         file (#778).
    ;   On Failure: Returns 0/
    ;  ERROR (optional, pass-by-reference) On failure returns a message.
    ;
    ;Required Setup: << Same as for A08^HLODEM1. >>
    ;
    N SEG,PARMS,WHOTO,HLA,HL
    K ERROR
    ;
    ;Build the message in the tradional manner in the HLA("HS") array.
    ;It requires that message delimiters be defined via HL("FS") and
    ;HL("ECH").
    S HL("FS")="|"
    S HL("ECH")="^~\&"
    D PID^HLODEM2(DFN,1,.SEG)
    S HLA("HLS",1)=SEG
    D NK1^HLODEM2(DFN,1,.SEG)
    S HLA("HLS",2)=SEG
    ;
    ;Now use the HLO APIs to send the message.
    S PARMS("MESSAGE TYPE")="ADT"
    S PARMS("EVENT")="AO8"
    I '$$NEWMSG^HLOAPI(.PARMS,.MSG,.ERROR) Q 0
    ;
    ;Move the message built in HLA("HLS") into MSG.
    D MOVEMSG^HLOAPI(.MSG,$NA(HLA("HLS")))
    ;
    S PARMS("SENDING APPLICATION")="HLO DEMO SENDING APPLICATION"
    S WHOTO("RECEIVING APPLICATION")="HLO DEMO RECEIVING APPLICATION"
    S WHOTO("FACILITY LINK NAME")="HLODEMO"
    Q $$SENDONE^HLOAPI1(.MSG,.PARMS,.WHOTO,.ERROR)
```

6.4 HLODEM4

```
HLODEM4 ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**146**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;** NOTE: The following segment and message builders
    ;are presented for demonstration purposes and are NOT fully
    ;developed.
    ;
A08(DFN,ERROR) --
    ;
    ;Description: Builds an ADT~A08 message and queues it for transmission
    ; using a sequence queue to guarantee the order of delivery. Included
    ; segments are the PID and NK1.
    ;
    ;Input:
    ;  DFN (required) The IEN of a patient record in the PATIENT file (#2)
    ;Output:
    ;  Function Return Value:
    ;   On Success: Returns the IEN of the messgage in the HLO MESSAGES
    ;    file (#778).
    ;   On Failure: Returns 0.
    ;  ERROR (optional, pass-by-refernce) On failure returns a message.
    ;
    ;Required Setup:
    ;<<HLO APPLICATION PARAMETER - file #779.2>>
    ; ** The sending application. **
    ; APPLICATION NAME: HLO DEMO SENDING APPLICATION
    ; ** Enter the routine to call that will alert folks that the sequence
    ;   queue is not moving. **
    ; SEQUENCE EXCEPTION TAG: LATE
    ; SEQUENCE EXCEPTION ROUTINE: HLODEM4
    ; ** Enter how many minutes to wait for an application acknowledgment
    ;   before generating an alert that something is wrong. **
    ; SEQUENCING TIMEOUT: 5
    ; ** Enter the package that is sending the message. **
    ; Package File Link: HL7 OPTIMIZED (HLO)
    ;
    ;<<HL Logical Link, file 870>>
    ; ** The receiving facility. **
    ; NODE: HLODEMO
    ; INSTITUTION: <The institution is entered here.>
    ; LLP TYPE: TCP
    ; MAILMAN DOMAIN: <The use of DNS DOMAIN is preferred for new links.>
    ; DNS DOMAIN: <The TCP/IP domain name for DNS is entered here.>
    ; TCP/IP ADDRESS: <The IP address is entered here.>
    ; TCP/IP SERVICE TYPE: CLIENT (SENDER)
    ; TCP/IP PORT (OPTIMIZED): <The port # is entered here.>
    ;
    ;
    N SEG,PARMS,WHOTO
    S PARMS("MESSAGE TYPE")="ADT"
    S PARMS("EVENT")="A08"
    I '$$NEWMSG^HLOAPI(.PARMS,.MSG,.ERROR) Q 0
    D PID^HLODEM1(DFN,1,.SEG)
    I '$$ADDSEG^HLOAPI(.MSG,.SEG,.ERROR) Q 0
    D NK1^HLODEM1(DFN,1,.SEG)
    I '$$ADDSEG^HLOAPI(.MSG,.SEG,.ERROR) Q 0
    ;
    ;
    S PARMS("SENDING APPLICATION")="HLO DEMO SENDING APPLICATION"
    ;
    ;** These parameters are mandatory if using a sequence queue. **
    ;The name of the sequence queue is arbitrary, but should be
    ;namespaced by the application.
    S PARMS("ACCEPT ACK TYPE")="AL"
    S PARMS("APP ACK TYPE")="AL"
    S PARMS("SEQUENCE QUEUE")="HLO DEMO RECIEVING APPLICATION" ;named sequen
    ce queue
    ;
    S WHOTO("RECEIVING APPLICATION")="HLO DEMO RECEIVING APPLICATION"
    S WHOTO("FACILITY LINK NAME")="HLODEMO"
    Q $$SENDONE^HLOAPI1(.MSG,.PARMS,.WHOTO,.ERROR)
    ;
LATE  ;The sending application secified this routine in the entry in the
    ;HLO APPLICATION REGISTRY file #779.2 for HLO DEMO SENDING
    ;APPLICATION. (see above)
    ;
    ;Enter code that will alert the operations staff in the event that the
    ;sequence queue stalls. It could be an email message or other
    ;communication.
    ;
    Q
```

6.5 HLODEM5

```
HLODEM5 ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**146**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;
PARSEMSG --
    ;
    ;Description: This is the default message handler for the receiving
    ; application named HLO DEMO RECEIVING APPLICATION.
    ;
    ;Input:
    ;  HLMSGIEN: At the point HLO calls this message handler, the variable
    ;       HLMSGIEN is set to the IEN of the message in the HLO
    ;       MESSAGE ADMINISTRATION file, #779.2.
    ;
    ;Output: none
    ;
    ;Required Setup:
    ;<<HLO APPLICATION PARAMETER - file #779.2>>
    ; ** The receiving application. **
    ; APPLICATION NAME: HLO DEMO RECEIVING APPLICATION
    ; ** The application chose to specify a private queue for its incoming
    ;   messages. If not specified, the queue 'DEFAULT' would be used. **
    ; DEFAULT PRIVATE IN-QUEUE: HLO DEMO RECEIVING APPLICATION
```

```
     ; **The application did not specify a message handler for the ADT~A08
     ;  message, but did specify a default handler. **
     ;  DEFAULT ACTION TAG: PARSEMSG
     ;  DEFAULT ACTION ROUTINE: HLODEM5
     ; ** The package that is receiving the message. **
     ; Package File Link: HL7 OPTIMIZED (HLO)
     ;
     N MSG,HDR
     I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.HDR) D ERROR("lost message") QUIT
     I 'MSG("BATCH"),HDR("MESSAGE TYPE")="ADT",HDR("EVENT")="A08" D
     .D A08(.MSG)
     .I HDR("APP ACK TYPE")="AL" ;(Need to respond with an application ack.)
     E D ERROR("unexpected message type") QUIT
     Q
     ;
PARSEA08 --
     ;
     ;Description:
     ; This is the ADT~A08 message handler for the receiving application
     ; named HLO DEMO RECEIVING APPLICATION.
     ;Input:
     ; At the point it is called, the variable HLMSGIEN is set to the IEN
     ; of the message in the HLO MESSAGE ADMINISTRATION file, #779.2.
     ;
     ;Required Setup:
     ;<<HLO APPLICATION PARAMETER - file #779.2>>
     ; ** The receiving application. **
     ; APPLICATION NAME: HLO DEMO RECEIVING APPLICATION
     ; **The application specified a specific message handler. **
     ; HL7 MESSAGE TYPE: ADT
     ; HL7 EVENT: A08
     ; ACTION TAG: PARSEA08
     ; ACTION ROUTINE: HLODEM5
     ; ** The package that is receiving the message. **
     ; Package File Link: HL7 OPTIMIZED (HLO)
     ;
     ;
     N MSG,HDR,PID,NK1
     I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.HDR) D ERROR("lost message") QUIT
     ;
     ;The message type and event are already known to be ADT~A08 based on the
     ;HLO Application Registry, but they could be checked by looking at
     ;MSG("MESSAGE TYPE") and MSG("EVENT")
     ;
     ;Parse the remaining segments of the message.
     D A08(.MSG,.PID,.NK1)
     ;
     ;
     ;(At this point the message's data has been retrieved into the PID and
     ; NK1 arrays. Process it! Then return an application acknowledgment
     ; if requested.)
     ;
     Q
     ;
A08(MSG,PID,NK1) --
     ;
     ;Description: $$STARTMSG^HLOPRS was already called. Parse the remaining
```

```
    ;       segments.
    ;Input:
    ; MSG (required, pass-by-reference) The message array returned by
    ;   callng $$STARTMSG^HLOPRS.
    ;Output:
    ; PID (pass-by-reference) The fields from the PID segment are returned
    ;    in this array.
    ; NK1 (pass-by-reference) The fields parsed from the NK1 segment are
    ;    returned in this array.
    ;
    N SEG
    F Q:'$$NEXTSEG^HLOPRS(.MSG,.SEG) D
    .;Base the parsing on the type of segment.
    .;SEG(0) is equivalent to SEG("SEGMENT TYPE")
    .I SEG("SEGMENT TYPE")="PID" D PID(.SEG,.PID) Q
    .I SEG(0)="NK1" D PID(.SEG,.PID) Q
    .;If not a PID or NK1 segment, disregard it.
    ;
    Q
    ;
PID(SEG,PID) --
    ;
    ;Description:
    ; Put the needed data into the PID array.
    ;Input:
    ; SEG (pass-by-reference) The parsed segment.
    ;Output:
    ; NK1 (pass-by-reference) Will return the specific data needed by the a
    pplication.
    ;
    N I,VALUE
    K PID
    ;Get the patient identifiers from the repeating field.
    F I=1:1 S VALUE=$$GET^HLOPRS(.SEG,3,4,1,I) Q:VALUE="" D
    .I VALUE="USVHA" S PID("ICN")=$$GET^HLOPRS(.SEG,3,1,1,I)
    .I VALUE="USSSA" S PID("SSN")=$$GET^HLOPRS(.SEG,3,1,1,I)
    ;
    ;Get the patient name.
    D GETXPN^HLOPRS2(.SEG,.VALUE,2)
    M PID("NAME")=VALUE
    ;
    ;Get the patient DOB.
    D GETDT^HLOPRS2(.SEG,.VALUE,7)
    S PID("DOB")=VALUE
    ;
    ;Get the patient address.
    D GETAD^HLOPRS2(.SEG,.VALUE,11)
    M PID("ADDRESS")=VALUE
    ;
    Q
NK1(SEG,NK1) --
    ;
    ;Description: Returns the needed data from the NK1 segment.
    ;Input:
    ; SEG (pass-by-reference) The parsed segment.
    ;Output:
    ; NK1 (pass-by-reference) Will return the specific data needed by the
    ;   application.
    N VALUE
    K NK1
    ;
    ;Get the contact's name.
    D GETXPN^HLOPRS2(.SEG,.VALUE,2)
    M NK1("NAME")=VALUE
    ;
    ;Get the contact's relationship.
    S NK1("RELATIONSHIP")=$$GET^HLOPRS(.SEG,3,2)
    ;
    ;Get the contact's address.
    D GETAD^HLOPRS2(.SEG,.VALUE,4)
    M NK1("ADDRESS")=VALUE
    ;
    ;Get the contact's phone number.
    S NK1("PHONE")=$$GET^HLOPRS(.SEG,5)
    ;
    ;get contact's role the contact
    S NK1("ROLE")=$$GET^HLOPRS(.SEG,7,2)
    Q
    ;
ERROR(ERROR) --
    ;report error
    ;{needs to be coded}
    Q
```

6.6 HLODEM6

```
HLODEM6 ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**146**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;
PARSEA08 --
    ;
    ;Description: This is the ADT~A08 message handler for the receiving
    ; application named HLO DEMO RECEIVING APPLICATION. This example uses
    ; hardcoded segment parsers.
    ;
    ;Input:
    ;  At the point it is called, the variable HLMSGIEN is set to the IEN
    ;  of the message in the HLO MESSAGE ADMINISTRATION file, #779.2.
    ;
    ;Required Setup:
    ;<<HLO APPLICATION PARAMETER - file #779.2>>
    ; ** The receiving application. **
    ; APPLICATION NAME: HLO DEMO RECEIVING APPLICATION
    ; **The application specified a specific message handler. **
    ; HL7 MESSAGE TYPE: ADT
    ; HL7 EVENT: A08
    ; ACTION TAG: PARSEA08
    ; ACTION ROUTINE: HLODEM6
    ; ** The package that is receiving the message. **
    ; Package File Link: HL7 OPTIMIZED (HLO)
    ;
    ;
    N MSG,HDR,SEG,PID,NK1
    I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.HDR) D ERROR("lost message") QUIT
    ;
    ;The message type and event are already known because of how the
    ;receving application was setup in the HLO Application Registry.
    ;
    ;Set up the delimiters in HL array required by the segment parsers.
    N HL
    S HL("FS")=HDR("FIELD SEPARATOR")
    S HL("ECH")=HDR("ENCODING CHARACTERS")
    ;
    ;loop through the segments
    F Q:'$$HLNEXT^HLOMSG(.MSG,.SEG) D
    .I $E(SEG(1),1,3)="PID" D PID(SEG(1),.PID)
    .I $E(SEG(1),1,3)="NK1" D NK1(SEG(1),.NK1)
    .;If not of these segment types, ignore it.
    ;
    ;(At this point the message's data has been retrieved into the PID and
    ; NK1 arrays. Process it! Then return an application acknowledgment
    ; if requested - not shown here.)
    Q
    ;
PID(SEG,PID) --
    ;
    ;Description: A hardcoded parser for the PID segment.
    ;Input:
    ; SEG The un-parsed segment.
    ; HL("FS"),HL("ECH") should be defined.(message delimiters)
    ;Output:
    ; NK1 (pass-by-reference) Will return the specific data needed by the a
    pplication.
    ;
    ; **WARNING**: The following code makes two WRONG assumptions:
    ;  1) That the data does not include escape sequences for delimiters.
    ;  2) That the entire segment is contained on a single node.
    ;
    N I,VALUE,FS,CS,RS,SS
    K PID
    ;
    ;For convenience, set the essage delimiters into variables.
    S FS=HL("FS") ;field separator
    S CS=$E(HL("ECH"),1) ;component separator
    S RS=$E(HL("ECH"),2) ;repetition separator
    S SS=$E(HL("ECH"),4) ;subcomponent separator
    ;
    ;Get the patient identifiers from the repeating field.
    F I=1:1 S VALUE=$P($P($P($P(SEG,FS,4),RS,I),CS,4),SS,1) Q:VALUE="" D
    .I VALUE="USVHA" S PID("ICN")=$P($P($P($P(SEG,FS,4),RS,I),CS,1),SS,1)
    .I VALUE="USSSA" S PID("SSN")=$P($P($P($P(SEG,FS,4),RS,I),CS,1),SS,1)
    ;
    ;Get the patient name.
    S PID("NAME","FAMILY")=$P($P($P($P(SEG,FS,3),RS,1),CS,1),SS,1)
    S PID("NAME","GIVEN")=$P($P($P($P(SEG,FS,3),RS,1),CS,2),SS,1)
    S PID("NAME","SECOND")=$P($P($P($P(SEG,FS,3),RS,1),CS,3),SS,1)
    S PID("NAME","SUFFIX")=$P($P($P($P(SEG,FS,3),RS,1),CS,4),SS,1)
```

```
     S PID("NAME","PREFIX")=$P($P($P($P(SEG,FS,3),RS,1),CS,5),SS,1)
     S PID("NAME","DEGREE")=$P($P($P($P(SEG,FS,3),RS,1),CS,6),SS,1)
     ;
     ;Get the patient DOB.
     S PID("DOB")=$$HL7TFM^XLFDT($P($P($P($P(SEG,FS,8),RS,1),CS,1),SS,1))
     ;
     ;Get the patient address.
     S PID("ADDRESS","STREET1")=$P($P($P($P(SEG,FS,12),RS,1),CS,1),SS,1)
     S PID("ADDRESS","STREET2")=$P($P($P($P(SEG,FS,12),RS,1),CS,2),SS,1)
     S PID("ADDRESS","CITY")=$P($P($P($P(SEG,FS,12),RS,1),CS,3),SS,1)
     S PID("ADDRESS","STATE")=$P($P($P($P(SEG,FS,12),RS,1),CS,4),SS,1)
     S PID("ADDRESS","ZIP")=$P($P($P($P(SEG,FS,12),RS,1),CS,5),SS,1)
     S PID("ADDRESS","COUNTRY")=$P($P($P($P(SEG,FS,12),RS,1),CS,6),SS,1)
     S PID("ADDRESS","TYPE")=$P($P($P($P(SEG,FS,12),RS,1),CS,7),SS,1)
     S PID("ADDRESS","OTHER")=$P($P($P($P(SEG,FS,12),RS,1),CS,8),SS,1)
     ;
     Q
     ;
NK1(SEG,NK1) --
     ;
     ;Description: A hardcoded parser for the NK1 segment.
     ;Input:
     ; SEG -the segment
     ; HL("FS"),HL("ECH") should be defined.(message delimiters)
     ;Output:
     ; NK1 (pass-by-reference) Will return the specific data needed by the
     ;   a pplication.
     ;
     ; **WARNING**: The following code makes two WRONG assumptions:
     ;  1) That the data does not include escape sequences for delimiters.
     ;  2) That the entire segment is contained on a single node.
     ;
     N VALUE,FS,CS,RS,SS
     K NK1
     ;
     ;For convenience, set the essage delimiters into variables.
     S FS=HL("FS") ;field separator
     S CS=$E(HL("ECH"),1) ;component separator
     S RS=$E(HL("ECH"),2) ;repetition separator
     S SS=$E(HL("ECH"),4) ;subcomponent separator
     ;
     ;Get contact's name.
     S NK1("NAME","FAMILY")=$P($P($P($P(SEG,FS,3),RS,1),CS,1),SS,1)
     S NK1("NAME","GIVEN")=$P($P($P($P(SEG,FS,3),RS,1),CS,2),SS,1)
     S NK1("NAME","SECOND")=$P($P($P($P(SEG,FS,3),RS,1),CS,3),SS,1)
     S NK1("NAME","SUFFIX")=$P($P($P($P(SEG,FS,3),RS,1),CS,4),SS,1)
     S NK1("NAME","PREFIX")=$P($P($P($P(SEG,FS,3),RS,1),CS,5),SS,1)
     S NK1("NAME","DEGREE")=$P($P($P($P(SEG,FS,3),RS,1),CS,6),SS,1)
     ;
     ;Get contact's relationship.
     S NK1("RELATIONSHIP")=$P($P($P($P(SEG,FS,4),RS,1),CS,2),SS,1)
     ;
     ;Get contact's address.
     S NK1("ADDRESS","STREET1")=$P($P($P($P(SEG,FS,5),RS,1),CS,1),SS,1)
     S NK1("ADDRESS","STREET2")=$P($P($P($P(SEG,FS,5),RS,1),CS,2),SS,1)
     S NK1("ADDRESS","CITY")=$P($P($P($P(SEG,FS,5),RS,1),CS,3),SS,1)
     S NK1("ADDRESS","STATE")=$P($P($P($P(SEG,FS,5),RS,1),CS,4),SS,1)
```

```
    S NK1("ADDRESS","ZIP")=$P($P($P($P(SEG,FS,5),RS,1),CS,5),SS,1)
    S NK1("ADDRESS","COUNTRY")=$P($P($P($P(SEG,FS,5),RS,1),CS,6),SS,1)
    S NK1("ADDRESS","TYPE")=$P($P($P($P(SEG,FS,5),RS,1),CS,7),SS,1)
    S NK1("ADDRESS","OTHER")=$P($P($P($P(SEG,FS,5),RS,1),CS,8),SS,1)
    ;
    ;Get contact's phone.
    S NK1("PHONE")=$P($P($P($P(SEG,FS,6),RS,1),CS,2),SS,1)
    ;
    ;Get contact's role the contact.
    S NK1("ROLE")=$P($P($P($P(SEG,FS,8),RS,1),CS,2),SS,1)
    Q
    ;
ERROR(ERROR) --
    ;report error
    ;(Needs to be coded.)
    Q
```

6.7 HLODEM7

```
HLODEM7 ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**144**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;
BATCH ;
    ;Description: This is the batch message handler for the receiving
    ; application named HLO DEMO RECEIVING APPLICATION. It parses
    ; the ADT~A08 messages created in BATCHA08^HLODEM1, though a batch of
    ; messages is allowed to contain different types of messages.
    ;
    ;Input:
    ;  At the point it is called, the variable HLMSGIEN should be set to
    ;  the IEN of the message in the HLO MESSAGE ADMINISTRATION file,
    ;  #779.2.
    ;Output: none
    ;
    ;Required Setup:
    ;<<HLO APPLICATION PARAMETER - file #779.2>>
    ; ** The receiving application. **
    ; APPLICATION NAME: HLO DEMO RECEIVING APPLCATION
    ; **The application specified a batch message handler. **
    ; BATCH ACTION TAG: BATCH
    ; BATCH ACTION ROUTINE: HLODEM7
    ; ** The package that is receiving the message. **
    ; Package File Link: HL7 OPTIMIZED (HLO)
    ;
    ;
    ;Create the variable workspace.
    ;
    N MSG,BHS,MSH
    ;
    ;Start the parsing.
    ;
    I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.BHS) D ERROR("lost message") QUIT
    ;
    ;The application mayverify that this is indeed a batch of messages by
    ;checking MSG("BATCH"). The check is not necessary if the
    ;receiving application setup is correct.
    ;
    I 'MSG("BATCH") D ERROR("not a batch message") QUIT
    ;
    ;Step through each message in the batch and process it.
    ;
    F Q:'$$NEXTMSG^HLOPRS(.MSG,.MSH) D
    .;
    .;If it is not known in advance what type of messages the batch
    .;contains then determine that from the message header.
    .I MSH("MESSGE TYPE")="ADT",MSH("EVENT")="A08" D
    ..;
    ..N PID,NK1
    ..;
    ..;Step through the segments of the message.
    ..F Q:'$$NEXTSEG^HLOPRS(.MSG,.SEG) D
    ...;
    ... ;Look at the segment type, then get its data.
    ...I SEG("SEGMENT TYPE")="PID" D PID^HLODEM5(.SEG,.PID)
    ...I SEG("SEGMENT TYPE")="NK1" D NK1^HLODEM5(.SEG,.NK1)
    ..;
    ..;(All the data has been parsed from the message. Now process it -
    ..; not shown.)
    Q
    ;
ERROR(ERROR) --
    ;reports an exception
    ;(Needs to be coded.)
    Q
```

6.8 HLODEM8

```
HLODEM8 ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**144**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;
APPACK ;Description:
    ; This is for receiving application acknowledgments. In
    ; this example, it just reports errors to support staff and sets
    ; the purge date of the original message to one month in the future.
    ;
    ;Input:
    ; At the point it is called, the variable HLMSGIEN is set to the IEN
    ; of the message in the HLO MESSAGE ADMINISTRATION file, #779.2.
    ;
    ;Required Setup:
    ; If this routine was designated as the callback routine when
    ; the original message was generated then no additional setup is
    ; needed. An alternative to using callbacks is to set up the message
    ; type & event for the acknowledgment in the HLO Application Registry.
    ;
    N MSG,HDR,SEG
    ;
    ;start parsing the acknowledgment.
    I $$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.HDR) D
    .;step through the segments looking for the MSA segment.
    .F Q:'$$NEXTSEG^HLOPRS(.MSG,.SEG) D
    ..I SEG("SEGMENT TYPE")="MSA" D
    ...;Check MSA-1 acknowledgment code. If not AA it is an error.
    ...I $$GET^HLOPRS(.SEG,1)'="AA" D
    ....;Reset the purge date of the original message to 1 month in future.
    ....N TIME S TIME=$$FMADD^XLFDT($$NOW^XLFDT,30)
    ....I '$$SETPURGE^HLOAPI3(MSG("ACK TO IEN"),TIME)
    ....;Send the staff an alert.
    ....;MSA-2 is the message id of the original message
    ....;MSA-3 is the error text
    ....D ALERT($$GET^HLOPRS(.SEG,2),$$GET^HLOPRS(.SEG,3))
    Q
    ;
ALERT(MSGID,ERROR) --
    ;alert support staff to the error
    ;(Needs to be coded.)
    Q
```

6.9 HLODEM9

```
HLODEM9 ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**144**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;
PARSEA08 --
    ;
    ;Description:
    ; This is the ADT~A08 message handler for the receiving application
    ; named HLO DEMO CLIENT. It is the same as PARSEA08^HLODEM5 except
    ; that it has been enhanced to return an application acknowledgment.
    ;
    ;Input:
    ;  At the point this is called, the variable HLMSGIEN is set to the IEN
    ; of the message in the HLO MESSAGE ADMINISTRATION file, #779.2.
    ;
    ;Required Setup:
    ;<<HLO APPLICATION PARAMETER - file #779.2>>
    ; ** The receiving application. **
    ; APPLICATION NAME: HLO DEMO CLIENT
    ; **The application specified a specific message handler. **
    ; HL7 MESSAGE TYPE: ADT
    ; HL7 EVENT: A08
    ; ACTION TAG: PARSEA08
    ; ACTION ROUTINE: HLODEM5
    ; ** the package that is receiving the message. **
    ; Package File Link: HL7 OPTIMIZED (HLO)
    ;
    ;
    N MSG,HDR,PID,NK1
    I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.HDR) D ERROR("lost message") QUIT
    ;
    ;The message type and event are already known to be ADT~A08 based
    ;on the setup. Parse the PID and NK1 segments.
    D A08^HLODEM5(.MSG,.PID,.NK1)
    ;
    ;Was an application acknowlegment requested?
    I HDR("APP ACK TYPE")="AL" D
    .;
    .;
    .;If the PID had a social security number return AA.
    .I $L(PID("SSN"))=9
    Q
    ;
ERROR(ERROR) --
    ;report error
    ;(Needs to be coded.)
    Q
```

6.10 HLODEM10

```
HLODEM10 --
    ;ALB/CJM-HL7 - Demonstration Code ;04/20/2009
    ;;1.6;HEALTH LEVEL SEVEN;**144**;Oct 13, 1995;Build 34
    ;Per VHA Directive 2004-038, this routine should not be modified.
    ;
    ;
    ;
PARSEA08 --
    ;
    ;Description:
    ; This is the ADT~A08 message handler for the receiving application
    ; named HLO DEMO RECEIVING APPLICATION. It is identical to
    ; PARSEA08^HLODEM5, except that it additionally returns an applicaiton
    ; acknowledgment.
    ;
    ;Input:
    ; At the point it is called, the variable HLMSGIEN is set to the IEN
    ; of the message in the HLO MESSAGE ADMINISTRATION file, #779.2.
    ;
    ;Output: none
    ;
    ;Required Setup: see PARSEA08^HLODEM5
    ;
    N MSG,MSH,PID,NK1
    I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.MSH) D ERROR("lost message") Q
    ;
    ;Parse the remaining segments of the message.
    D A08^HLODEM5(.MSG,.PID,.NK1)
    ;
    ;
    ;{At this point the message's data has been retrieved into the PID and
    ; NK1 arrays. Process it - not shown!}
    ;
    ;If an application acknowledgment was requested, then return one.
    I MSH("APP ACK TYPE")="AL" D
```

```
     .N ACK,PARMS
     .;return AA if the PID had a social security number,or AE=error if not
     .I $L($G(PID("SSN")))=9 D
     ..S PARMS("ACK CODE")="AA"
     .E D
     ..S PARMS("ACK CODE")="AE",PARMS("ERROR MESSAGE")="MISSING SSN"
     .I '$$ACK^HLOAPI2(.MSG,.PARMS,.ACK,.ERROR) D ERROR("$$ACK^HLOAPI2 FAILED
     ") Q
     .;
     .;There are no additional segments to add to the acknowledgment, so
     .;send it.
     .I '$$SENDACK^HLOAPI2(.ACK,.ERROR) D ERROR(ERROR)
     ;
     Q
     ;
BATCH ;Description:
     ; This is an extension of the batch message handler in BATCH^HLODEM7.
     ; In addition to parsing a batch of messages as in BATCH^HLODEM7, it
     ; also returns a batch of application acknowledgments.
     ;
     ;Input:
     ;  At the point it is called, the variable HLMSGIEN should be set to
     ;  the IEN of the message in the HLO MESSAGE ADMINISTRATION file,
     ;  #779.2.
     ;
     ;Output: none
     ;
     ;Required Setup: see BATCH^HLODEM7
     ;
     N MSG,BHS,MSH,ACK
     ;
     ;Start the parsing.
     I '$$STARTMSG^HLOPRS(.MSG,HLMSGIEN,.BHS) D ERROR("lost message") QUIT
     ;
     ;Is a return batch of application acknowledgments requested?
     ;If so, start the return batch.
     I BHS("APP ACK TYPE")="AL",'$$BATCHACK^HLOAPI3(.MSG,,.ACK,.ERROR) D ERRO
     R(ERROR)
     ;
     ;Step through each message in the batch and process it.
     F Q:'$$NEXTMSG^HLOPRS(.MSG,.MSH) D
     .;
     .;(If it is not known in advance what type of messages the batch
     .;contains then determine that from the message header. But here
     .; it is assumed to be an ADT~A08.)
     .
     .;Parse the remaining segments of the message.
     .D A08^HLODEM5(.MSG,.PID,.NK1)
     .;
     .;(All the data has been parsed from the message. Now process it! Not
     .; shown.)
     .;
     .;Might check the individual message to determine if an acknowledgment
     .;was requested. Whether or not this is necessary depends on the
     .;negotiated interface specification, but usually it is NOT necessary.
     .I MSH("APP ACK TYPE")="AL" D
     ..N PARMS
```

```
    ..;return AA if the PID had a social security number,or AE=error if not
    ..I $L($G(PID("SSN")))=9 D
    ...S PARMS("ACK CODE")="AA"
    ..E D
    ...S PARMS("ACK CODE")="AE",PARMS("ERROR MESSAGE")="MISSING SSN"
    ..;
    ..;Add a message to the return batch.
    ..I '$$ADDACK^HLOAPI3(.ACK,.PARMS,.ERROR) D ERROR(.ERROR)
    .;
    .;There are no additional messages to add to the return batch, so
    .;send it.
    .I '$$SENDACK^HLOAPI2(.ACK,.ERROR) D ERROR(ERROR)
    Q
    ;
ERROR(ERROR) --
    ;report error
    ;(Needs to be coded.)
    Q
```

7.0 Quick Overview of the HLO User Interface

7.1 HLO Main Menu

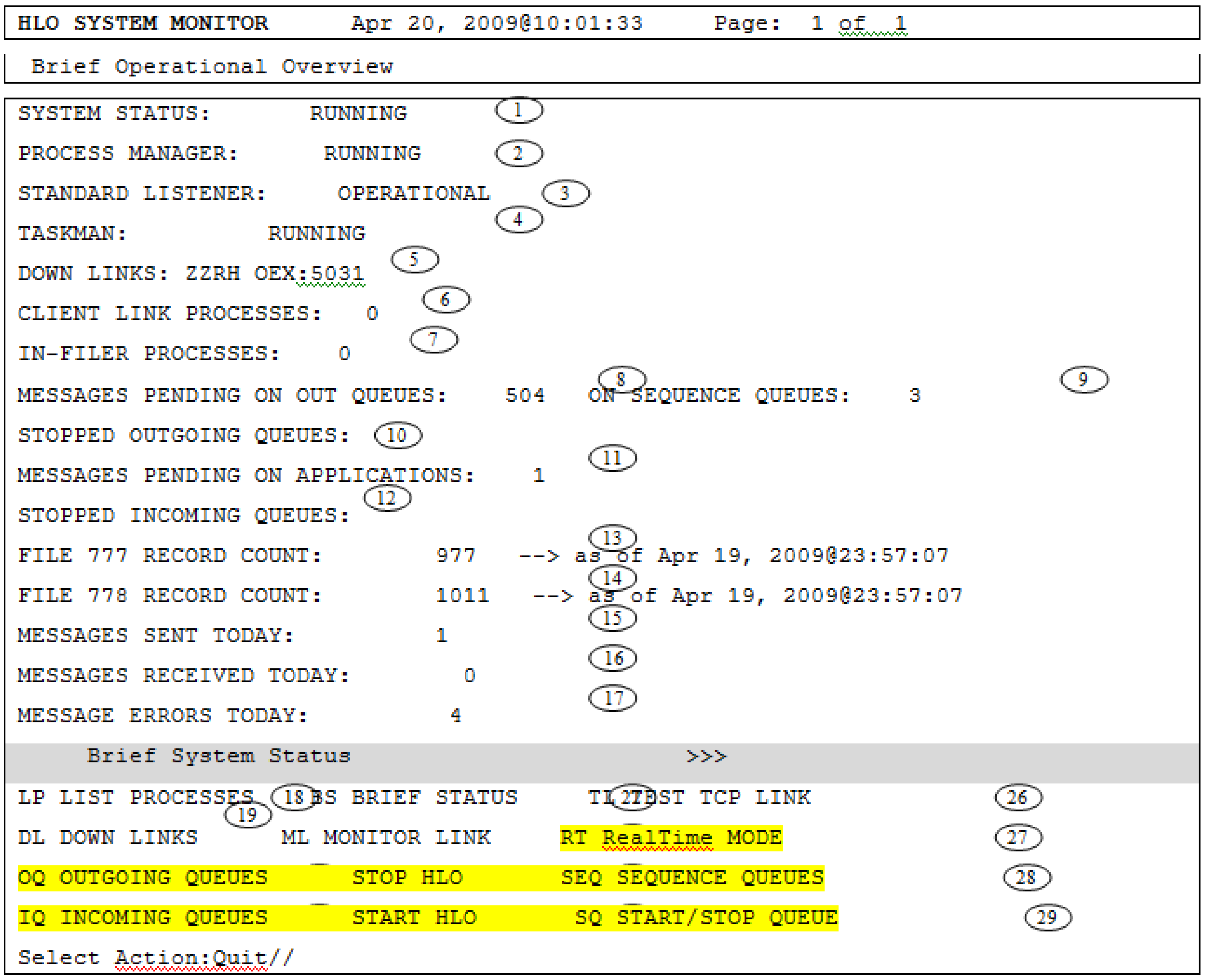
This is the top-level menu for HLO:

```
  SM   HLO SYSTEM MONITOR
  MV   HLO MESSAGE VIEWER
  STAT  HLO MESSAGE STATISTICS
  ES   HLO ERROR STATISTICS
  DM   HLO DEVELOPER MENU ...
  SP   EDIT HLO SYSTEM PARAMETERS

Select HL7 (Optimized) MAIN MENU Option:
```

- The HLO System Monitor is used for controlling and monitoring the operation of HLO. It is described in more detail below.
- The HLO Message Viewer is the main tool for viewing messages. Its features include:
    - The ability to search for messages by date range, application, and message type.
    - Reporting of message errors by date range and application.
    - It displays messages, using the message ID from the message header as the lookup key. It also shows all the administrative data related to the message, such as the status and date/time transmitted. It also links messages to their application acknowledgments and visa versa. Commit acknowledgments are stored with the original message and can only be accessed through the message id of the original message.
- The HLO Message Statistics option provides reporting capability for counts of messages sent and received over any time period. Monthly, daily, and hourly statistics are maintained.
- The HLO Error Statistics option provides reporting capability for counts of message errors over any time period. Monthly, daily, and hourly statistics are maintained.
- The HLO Developer Menu contains options that are needed to develop messaging applications with HLO. They have been described elsewhere in this manual.
- The Edit HLO System Parameters option is used by system managers to configure their system. It contains parameters used by HLO for sending, receiving, and purging messages.

7.2 HLO System Monitor



```
HLO SYSTEM MONITOR       Apr 20, 2009@10:01:33      Page:  1 of  1
 Brief Operational Overview
SYSTEM STATUS:        RUNNING
PROCESS MANAGER:       RUNNING
STANDARD LISTENER:      OPERATIONAL
TASKMAN:             RUNNING
DOWN LINKS: ZZRH OEX:5031
CLIENT LINK PROCESSES:    0
IN-FILER PROCESSES:     0
MESSAGES PENDING ON OUT QUEUES:    504    ON SEQUENCE QUEUES:    3
STOPPED OUTGOING QUEUES:
MESSAGES PENDING ON APPLICATIONS:    1
STOPPED INCOMING QUEUES:
FILE 777 RECORD COUNT:        977    --> as of Apr 19, 2009@23:57:07
FILE 778 RECORD COUNT:        1011    --> as of Apr 19, 2009@23:57:07
MESSAGES SENT TODAY:          1
MESSAGES RECEIVED TODAY:        0
MESSAGE ERRORS TODAY:          4
     Brief System Status                        >>>
LP LIST PROCESSES    BS BRIEF STATUS     TT TEST TCP LINK
DL DOWN LINKS     ML MONITOR LINK     RT RealTime MODE
OQ OUTGOING QUEUES       STOP HLO      SEQ SEQUENCE QUEUES
IQ INCOMING QUEUES       START HLO      SQ START/STOP QUEUE
Select Action:Quit//
```

1. Shows whether HLO is running.
2. Shows whether the HLO Process Manager is running. The HLO Process Manager is a process that starts and stops other HLO processes as needed, and can respond automatically to changes in the workload. The HLO Process Manager should always be running while HLO is running. If it stops unexpectedly, HLO may continue to run, but no new HLO processes will be started.
3. Shows whether the HLO listener is running. In VHA the standard listener is a VMS TCPIP Service running on a standard port.
4. Shows whether Taskman is running.
5. Shows a list of the links that have recently been failing.
6. Shows a count of the number of client proesses that are running. Client processes are responsible for transmitting the messages that are pending on the outgoing queues.
7. Shows a count of the in-filer processes that are running. The in-filer processes are responsible for passing newly received messages to the receiving application.

8. Shows a count of the messages pending to be transmitted.
9. Shows a count of the messages pending on sequence queues. Sequence queues are used to guarantee the order of message delivery.
10. Shows a list of outgoing queues that have been stopped via the START/STOP QUEUE action.
11. Shows a count of newly received messages that are still waiting to be passed to the application by the in-filer process.
12. Shows a list of the incoming queues that have been stopped via the START/STOP QUEUE action.
13. Shows the most recent count of the records in the HLO MESSAGE BODY file (#777). Since FileMan is not used to add or delete records to this file, the 0 node of the file is not updated with a count of the records each time a record is added or deleted. Instead, there is a special option that runs several times during the day that counts the records.
14. Shows the most recent count of the records in the HLO MESSAGES file (#778). Since FileMan is not used to add or delete records to this file, the 0 node of the file is not updated with a count of the records each time a record is added or deleted. Instead, there is a special option that runs several times during the day that counts the records.
15. Shows the count of all messages sent today. Commit acknowledgments aren’t included in the count.
16. Shows a count of the messages received today. Commit acknowledgments aren’t included n the count.
17. Shows a count of the messages whose status has been set to ERROR today.
18. This action will show a list of the HLO processes that are currently running or scheduled to run.
19. This action will display a screen that lists the links that recently have been failing. It has actions for manually starting or stopping specific links.
20. This action will display a screen that lists all the outgoing queues that have messages pending on them. It has actions for deleting a queue and for deleting messages from queues.
21. This action will display a screen that lists all the incoming queues that have messages pending on them. It has actions for deleting a queue and for deleting messages from queues.
22. This action takes you to the screen that provides an overview of the current status of the HLO system. It is the same screen that is shown above.
23. This screen will show a near real-time display of the count of messages pending for transmission over a specific link.
24. This action is used to turn HLO off. It may take several minutes for all the processes to stop.
25. This action is used to turn HLO back on. It may take several minutes for all the HLO processes to be started via Taskman.
26. This action is used to test whether there is connectivity via TCP/IP over a specific link.
27. This action changes how the screen is displayed. In real-time mode the screen is updated every few seconds.
28. This action will display a screen that lists counts of messages pending on sequence queues. It has actions for deleting sequence queues and for advancing sequence queues.
29. This action is used to manually start or stop specific named queues. Stopping a named queue will cause all queues with that name to suspend transmission of its messages, regardless of the link over which the transmission is to take place.

8.0 Troubleshooting for the Developer

HLO provides two tools that are specifically for troubleshooting interface problems - the client trace and server trace tools described below. In addition, there are several general tools that may be useful when troubleshooting problems, including:

- The HLO SYSTEM MONITOR:
  - Has an action to test the connectivity to a remote link.
  - Provides the status of the HLO Client/Server engine.
  - Provides the status of incoming queues, outgoing queues, and sequence queues.
- The HLO MESSAGE VIEWER:
  - Provides message search capability.
  - Provides the ability to view messages and related administrative information.
  - Provides a visual parser for interpreting the content of messages.

8.1 The Client Trace Tool

The routine ^HLOTRACE is a utility for running the HLO client in the foreground. It asks the user to select a remote server to connect to, and at that point it attempts to transmit pending messages to that server. As it executes, it writes a stream of messages to the screen detailing each step of the process as it occurs.

To use the tool, there must be pending messages waiting for transmission. Furthermore, the tool has to be able to obtain a lock on the queue, so within the HLO System Monitor it is necessary to either:

- Shutdown the link for those messages by using the SHUTDOWN LINK action on the DOWN LINKS screen.
- Shutdown the specific queue by using the START/STOP QUEUE action.
- Shutdown the entire HLO system by using the STOP HLO action.

Example: Using the client trace tool.

First the ^HLOTRACE routine must be obtained and installed on the system. Then shut down either the specific queue, the link, or the entire HLO system. Then at the M prompt enter:

```
NXT>D ^HLOTRACE


Select a TCP link:  HLODEMO         Enter the client link.

What is the name of the queue: DEFAULT// Verify the queue.

Send how many at a time: (1-100): 1//  Enter number to trace.

Launching the client process

Trying to connect...

Connected!


Looking for the next message to transmit...

Message IEN=1461582 next on queue, do you want to trace its transmission? YES//

   Time: 3090407.085809  Beginning to transmit message....
   Time: 3090407.085809  Writing header segment...
MSH|^~\&|HLO DEMO SENDING APPLICATION|998^HL7.FO-OAKLAND.MED.VA.GOV:5011^DNS|HLO
 DEMO RECEIVING APPLICATION|612BY^TEST.DOMAIN:5031^DNS|20090407075126-0800||ADT^
AO8^|998 1461582|T^|2.4|||AL|NE|
   Time: 3090407.085809  Completed!
   Time: 3090407.085809  Writing next segment...
PID|1||3333^^^USVHA&&0363^NI~517509835^^^USSSA&&0363^SS||HOFFMAN^JOHN^^^^||19470
605||||4876 25TH AVE.^^SAN FRANCISCO^CALIFORNIA^94111^^^
   Time: 3090407.085809  Completed!
   Time: 3090407.085809  Writing next segment...
NK1|1|DOE^JOHN^^^^|^|^^^^^^^|||EP^EMERGENCY CONTACT PERSON^0131
   Time: 3090407.085809  Completed!
   Time: 3090407.085809  Writing message terminators and flushing buffer...
   Time: 3090407.085809  Completed!
   Time: 3090407.085809  Message transmitted!
   Time: 3090407.085809  Beginning to read commit acknowledgment....
   Time: 3090407.085809  Reading header...
   Time: 3090407.085809
MSH|^~\&|HLO DEMO RECEIVING APPLICATION|050^OEX.FO-OAKLAND.MED.VA.GOV^DNS|HLO DE
MO SENDING APPLICATION|998^HL7.FO-OAKLAND.MED.VA.GOV:5011^DNS|20090407085809-070
0||ACK|050 100000294967|T|2.4|||NE|NE
   Time: 3090407.085809  Completed!
   Time: 3090407.085809  Reading next segment...
   Time: 3090407.085809
MSA|CR|998 1461582|RECEIVING APPLICATION NOT DEFINED|
   Time: 3090407.085809  Completed!
   Time: 3090407.085809  Reading next segment...
   Time: 3090407.085809  No more segments!
   Time: 3090407.085809  Commit acknowledgment received!

Looking for the next message to transmit...

No more messages pending on that queue!
Cleaning up....
DONE!
```

8.2 The Server Trace Tool

The routine ^HLOSTRAC is a utility that runs the HLO server in the foreground. It asks the user to select a port to receive messages on, and then will attempt to read messages through that port. As the trace tool executes, it writes a stream of messages to the screen detailing each step of the process as it occurs.

The server trace tool must be able to open the port. If a server is already running on that port, that server must be stopped before using the server trace tool. If the server is a VMS TCPIP Service, the service must be stopped by using the VMS TCPIP 'DISABLE SERVICE' command.

Example: Using the server trace tool.

```
NXT>D ^HLOSTRAC


What port do you want to listen on while in server trace mode?
The port must be free. If a server already has it opened then the
server needs to be stopped before starting in server trace mode.
PORT:  (1-65535): 5011// 6666


Starting the server, hit the CTRL-C key to stop the server...



   Time: 3090810.121211   Opening the port...
   Time: 3090810.121211   Waiting for remote client to connect...
   Time: 3090810.121442   Remote client connected...
Beginning to read next message...
   Time: 3090810.121442   Reading message header...
   Time: 3090810.121442
MSH|^~\&|MYTEST|050^OEX.FO-OAKLAND.MED.VA.GOV:5031^DNS|MY_silly_TEST|^NXT.FO-OA|
   Time: 3090810.121442   Completed!
   Time: 3090810.121442   Parsing the message header...
   Time: 3090810.121442   Checking if duplicate message...
   Time: 3090810.121442   Reading next segment...
   Time: 3090810.121442
MSA||1
   Time: 3090810.121442   Completed!
   Time: 3090810.121442   Reading next segment...
   Time: 3090810.121442   No more segments!
```

```
Beginning to write the commit acknowledgment...
    Time: 3090810.121442   Writing header segment...
MSH|^~\&|MY_silly_TEST|998^NXT.FO-OAKLAND.MED.VA.GOV^DNS|MYTEST|050^OEX.FO-OAKLE
    Time: 3090810.121442   Completed!
    Time: 3090810.121442   Writing next segment...
MSA|CR|050 922237|RECEIVING APPLICATION NOT DEFINED|
    Time: 3090810.121442   Completed!
    Time: 3090810.121442   Writing message terminators and flushing buffer...
    Time: 3090810.121442   Completed!

Do you want to trace another message transmission? NO//
    Time: 3090810.121511   Error encountered, $ECODE=ZZHLOSTOP
        TCP connection was dropped
    Time: 3090810.121512   Closing the port...
NXT>
```