British Columbia
Professional and Software Conformance Standards

Electronic Health Information Exchange

Volume 5B: Transport Protocol – HNSecure

Version 5.1    2020-01-31

Security Classification: Low Sensitivity
Copyright Notice

Copyright © Province of British Columbia

All rights reserved.

This material is owned by the Government of British Columbia and protected by copyright law. It may not be reproduced or redistributed without the prior written permission of the Province of British Columbia.

Disclaimer and Limitation of Liabilities

This document and all of the information it contains is provided "as is" without warranty of any kind, whether express or implied.

All implied warranties, including, without limitation, implied warranties of merchantability, fitness for a particular purpose, and non-infringement, are hereby expressly disclaimed.

Under no circumstances will the Government of British Columbia be liable to any person or business entity for any direct, indirect, special, incidental, consequential, or other damages based on any use of this document, including, without limitation, any lost profits, business interruption, or loss of programs or information, even if the Government of British Columbia has been specifically advised of the possibility of such damages.

Document Details

Author: MOH Conformance and Integration Services
Date Created: 2004-08-05
Last Updated: 2020-01-31
Version: 5.1
# Table of Contents

1.0 Overview .......................................................................................................................... 5  
   1.1 Platform Considerations .............................................................................................. 5  
   1.2 Client Interfaces ......................................................................................................... 6  
      1.2.1 HNAPI ............................................................................................................... 7  
      1.2.2 HNClient ........................................................................................................... 7  
      1.2.3 HNLLI .............................................................................................................. 7  
   1.3 HNSERVER Interface ................................................................................................. 8  

2.0 HNAPI .................................................................................................................................. 9  
   2.1 Overview ..................................................................................................................... 9  
   2.2 Set-up ......................................................................................................................... 11  
   2.3 Start-up ...................................................................................................................... 11  
   2.4 Operation .................................................................................................................. 12  
   2.5 Communicating through HNAPI ............................................................................... 13  
   2.6 Connecting to HNCLIENT ......................................................................................... 14  
   2.7 Shutdown ................................................................................................................... 14  

3.0 HNCLIENT .......................................................................................................................... 15  
   3.1 Overview .................................................................................................................... 15  
   3.2 Set-up and Administration ......................................................................................... 18  
   3.3 Startup ....................................................................................................................... 18  
   3.4 Operation ................................................................................................................... 19  
      3.4.1 Client Application Interaction ............................................................................ 19  
      3.4.2 HNClient Communication Protocol ................................................................. 20  
   3.5 Shutdown .................................................................................................................... 22  

4.0 HNLLI ................................................................................................................................... 23  
   4.1 Overview ..................................................................................................................... 23  
   4.2 Set-up ......................................................................................................................... 24  
   4.3 Startup ....................................................................................................................... 25  
   4.4 Operation ................................................................................................................... 26  
      4.4.3 HNLLI Interface ................................................................................................. 26  
      4.4.4 Transaction Layer Function Types ...................................................................... 28  
      4.4.5 Connecting to HNGARD .................................................................................... 28  
   4.5 Shutdown .................................................................................................................... 29  

5.0 Security Services .............................................................................................................. 30  
   5.1 Public/Private Key Encryption .................................................................................... 31  
   5.2 DES Key Encryption .................................................................................................. 31  

6.0 Sample Applications ......................................................................................................... 32  
   6.1 Batch HL7 .................................................................................................................. 32  
   6.2 HNTIME ..................................................................................................................... 32  

7.0 HN6 Lower Level Protocol and Message Formats .......................................................... 33
7.1 Communications Protocol.......................................................................................33

8.0 HNSETUP ............................................................................................................. 35
  8.1 Overview ................................................................................................................35
  8.2 Set-up and Administration ......................................................................................36
  8.3 Startup ...................................................................................................................36
  8.4 Operation ...............................................................................................................37
  8.5 Public/Private Key Generation................................................................................38
  8.6 Shutdown ...............................................................................................................39

9.0 HNSERVER .......................................................................................................... 40
  9.1 HNSERVER Interface ............................................................................................ 40
  9.2 HNSERVER (Service) Overview ............................................................................41
  9.3 Server Application (Service) Overview ...................................................................42
     9.3.1 Basic Message Processing ........................................................................42
  9.4 HNSERVER Function .............................................................................................43
  9.5 Set-up ....................................................................................................................46
     9.5.1 Overview ....................................................................................................46
     9.5.2 Requirements ............................................................................................ 47
     9.5.3 Other Set-up Notes ....................................................................................49
  9.6 Startup ...................................................................................................................50
     9.6.4 Basic Client Connection Steps ................................................................... 51
     9.6.5 Basic Server Application Connection Steps ...............................................52
  9.7 Operation ...............................................................................................................53
     9.7.1 Server Application Interaction ....................................................................53
     9.7.2 HNSERVER Communication Protocol .......................................................54
  9.8 HNSERVER HL7 Transaction Routing ................................................................... 55
     9.8.1 File Attachments to HL7 Transactions .......................................................55
  9.9 Shutdown ...............................................................................................................56

10.0 HL7XFER .............................................................................................................. 57
  10.1 Overview ............................................................................................................57
  10.2 Communicating with an HL7XFER Listener ............................................................58
  10.3 Operation .............................................................................................................60
1.0 Overview

The HNSecure architecture is designed to support a network of computer-to-computer (i.e., application-to-application) connections where ‘clients’ can send requests (HL7 messages) and optionally, files to application ‘servers’ for processing or storage.

Transaction (message) security is implemented according to the HNSecure API, incorporating private/public keys and DES encryption. Application security is managed by a local system administrator.

The HNSecure software ensures only that the messages to be sent to the server originate from the specified client Facility ID and go to the specified server Facility ID in a secure manner. It also ensures that any responses from the remote server did in fact come from the remote server and that the message was not altered en route.

The HNSecure libraries and interfaces do not ensure that application programs using this software are in fact authorized to do so. They also do not verify that the person operating the application is authorized to do so.

The client library and interface options provide a single point of contact through which different client applications can gain access to Ministry of Health applications. HL7 requests (or transactions) are routed by the software, in a secure manner, to specified application servers which may be located on the same machine or, more likely, on remote 3rd party or ministry machines. The software maintains the connections to the server applications for the life of the request so that responses from the server are returned to the originating client. The connection is then broken. This cycle may be repeated any number of times.

The server interface provides a mechanism for a facility to listen for incoming messages, have them processed, and return the response back to the originating client.

1.1 Platform Considerations

The distributed client libraries and executables are targeted at Windows and AIX only. However, as much as possible, the software has been written with portability in mind. Almost all code modules are portable, written in ANSI C. Code modules that are non-portable have names that start with the letter ‘n’, for native, while portable code modules have names that start with the letter ‘p’. This decision was made to simplify ports to other operating systems.
1.2 Client Interfaces

The interfaces provided allow independently developed applications to communicate with the Ministry Health Information Exchange Systems (HIE). The interfaces provide connectivity and data request capabilities. They accept transactions (messages) from the client, routing them to the specified server application (local or remote).

There are three client application interface options through which all requests, files, data and responses flow to a destination server to be processed, stored and/or returned.

The shaded areas of the following diagram represent each of the different interface options available to application writers. Each will be described in later sections of this document.

Figure 1 HNClient Interfaces

The following lists each of these options from the highest level with the least flexibility but also lowest complexity to the lowest level with maximum flexibility and highest complexity:
1.2.1 HNAPI

A ‘C’ callable routine that accepts as input HL7 transactions and routes them through to the HNClient interface over sockets. The advantage of this method over HNClient is that applications do not need to be concerned about setting up and managing open sockets. It is multi-thread safe as well as multi-process safe meaning that any number of applications, each with any number of threads may attempt to communicate through the client library to individual remote servers. The only limitation is system constraints (e.g., limited number of sockets available, no multi-threading).

1.2.2 HNClient

A socket interface that provides a simple mechanism for applications to send HL7 transactions to a remote server. This interface uses standard TCP/IP sockets to transmit data and receive responses. This interface is designed to run as a service. It accepts requests from multiple applications on multiple threads throughout a LAN or from the local PC and passes them on to the HNClient API. The data sent through this interface is standard HL7 with block length indicators.

1.2.3 HNLLI

A low-level interface, this API is designed for custom applications which require a high degree of integration with the library or are working on a platform which precludes use of other solutions. This interface is also payload independent meaning that any type of data may be transmitted, not just HL7 messages. This interface ensures secure, reliable transport to remote servers.
1.3 HNSERVER Interface

The server interface provides a mechanism for a facility to listen for incoming messages, have them processed, and return the response back to the originating client.

The server interface is an optional component of the overall package. It is not necessary to set up the HNServer portion to use the Client Interface as described above.

The server interface employs the same security model as employed by the client portion of this package in that it authenticates facilities rather than the applications or the users employing them.

There is a single application interface (HNServer) for application developers to work with. This IP based interface is identical in format to the interface between HNAPI and HNClient. HNServer is discussed further in later sections of this document.

The HL7XFER library has been written and supplied with the distribution to aid in the communication between HNServer and the backend application.
2.0 HNAPI

2.1 Overview

HNAPI allows developers to link a very simple ‘C’ callable interface into their applications and have all of the connection, data formatting, transfer, socket management, security management, and other issues managed automatically.

The purpose of HNAPI is to accept HL7 transactions as input and pass them to an HNClient Service at a particular IP address listening on a particular IP port number. In response, an HL7 message is returned to the calling application.

The HNAPI Interface has no user level interface, as it is strictly a code library. Instead it is controlled through calls to the HNAPI function. This interface allows client applications to perform HL7 transactions with ministry servers including file transfers either to or from the local machine.

There is very little overhead on the part of the client application. It only needs to know how to formulate HL7 messages. The HNAPI routine uses TCP/IP \HOSTS and \SERVICES facilities to obtain the IP address and port number where the HNClient Service is listening.

One requirement of HNAPI is to have a valid TCP/IP stack on your system. This stack must be capable of communicating with other standard sockets. This allows for the possibility of using special third-party TCP/IP stacks in situations where standard WinSock or Berkeley Sockets are not available.

HNAPI is a thread safe library allowing a single application to have any number of pending transactions occurring simultaneously. The only limits are available system resources as well as the resources available to HNClient.
As the diagram below indicates, there are two modules central to the operation of HNAPI:

- **MSH Parser** - Verifies the contents of the HL7 MSH segment and fills in any optional fields it can which may have been omitted by the client application.

- **Socket Manager** - Manages the socket interface with HNClient for each thread simultaneously communicating with HNClient. It sends the HL7 data to HNClient and waits for a response.

Both modules are described in more detail later in this document.

*Figure 2 HNAPI operation*
2.2 Set-up

There are no specific set-up requirements for HNAPI other than an entry for HNClient in the HOSTS and SERVICES files. The HOSTS entry is only required if the HNClient service is listening on a different (possibly remote) IP address than 127.0.0.1. The SERVICES entry is only required if a port number other than the default, 19430, is to be used for local socket connections to HNClient.

Each targeted platform will require its own version of the HNAPI library and run-time support files. However, these support files are generally available as part of the compiler suite you are using to generate the library. The only requirement for these files is that they be placed in the installation directory, a system directory or some other directory in the system search path.

2.3 Start-up

As the design of HNAPI provides a mechanism which does not require explicit start up or shut down by client applications, the client may make immediate use of the library. Any initialization which needs to be performed is done automatically by the library itself either when the library is loaded and unloaded by the operating system, or through automated checks in the library code itself.

As the library is integrally linked into the client application, it is up to the client application to determine how shutdown is to occur. The client must decide the fate of pending transactions and file transfers.
2.4 Operation

Operation and interaction with the HNAPI library is performed strictly through a ‘C’ style call interface that is either dynamically or statically linked into the client application. This method provides maximum ease of use for clients to communicate with remote HNSecure servers.

Each client application process communicating through HNAPI may have any number of threads simultaneously sending HL7 transactions through the library. Each thread sends one message at a time. The only limitation is the availability of system resources on the local machine and the capabilities of the HNClient Service being communicated with.

Each transaction in the HNAPI library sent to HNClient is:

- atomic (completely processed or reversed, i.e., no action left dangling);
- stateless (does not depend on the results of previous transactions);
- confirmed (not complete until a response or error code is returned); and
- synchronous (the client thread waits for a response from the client before proceeding).

This implies that no conversations between the client and the remote server take place. A request is sent to the server and a reply is received. Each request is completely separate and independent of all others which take place before it. This does not mean however, that the client application cannot add context to the contents of a particular transaction.
2.5 Communicating through HNAPI

There are several steps that must be performed in order to send an HL7 transaction to a remote server:

1) Formulate the HL7 message to be transmitted. This message is in standard HL7 format (please refer to the Application Developer’s Guide document for more information regarding HL7 messages).

2) A valid TCP/IP stack must be initialized if not already available. In most environments, this normally occurs during system start up. In others, it may require intervention on the part of the application or user.

3) The HL7 Message plus an optional file name specification is passed to HNAPI. The thread waits for a response of some sort.

4) The response is returned, the application checks for error return codes. Note that this error return code is not the same as an HL7 error response. As far as this library is concerned, any HL7 response returned indicates a successful call to HNAPI.

5) If no error, the application parses and interprets the returned HL7 response. HL7 error responses are defined in the Application Developer’s Guide.

6) Any buffers used should be destroyed if they are not going to be used for a subsequent transaction.
2.6 Connecting to HNCLIENT

The following documents the process by which HNAPI connects to HNClient:

1) Parses the MSH Segment of the HL7 message for the domain.
2) Uses the domain to select an appropriate port number.
3) Using the IP address and port number, attempts a connection to HNClient.
4) If the connection attempt fails a connection failure HL7 message is returned to the client application and the transaction fails.

2.7 Shutdown

There are no specific requirements of HNAPI that must be performed prior to shut down as no state information or set-up conditions are recorded by this library.

Within the library, if specific modules do require internal initialization or shutdown, there will be specific functions named xxx_InitializeLibrary or xxx_ShutdownLibrary where xxx refers to a specific module. In addition, a flag is maintained which records the current status of the library (i.e., currently initialized, shutdown, requires initialization). All of this is automatically maintained internally to each module and requires no explicit outside set-up.
3.0 HNCLIENT

3.1 Overview

The HNClient application acts as a service to point of service applications (POS) providing a simple TCP/IP socket interface to secure HIE communications. HNClient itself is in fact a client application to the HNLLI library. It operates by listening for socket connection requests from client applications. Once accepted it spawns a thread to handle the connection then continues by listening for more socket connection requests from client applications.

Once accepted, HNClient handles all of the bookkeeping and interactions with the HNLLI library for the secure transmission of the supplied transaction with remote servers. The following diagram shows the functional dependencies of the HNClient service:

![HNClient service functional dependencies](image)

*Figure 3 HNClient service functional dependencies*
As can be seen in the above diagram, there are eight modules central to the operation of the HNClient service:

1) **IP Port Listener**

This module waits for client applications to attempt to communicate with the HNClient service. Once detected and accepted, it spawns a new thread (in systems which support multi-threading) to handle the transaction request and passes control to the HL7 Msg Identifier module.

This module makes use of the HL7XFER library to communicate HL7 messages and responses with client applications.

2) **HL7 Msg Identifier**

This module receives control at each accepted transfer request from the IP Port Listener module. It determines the type of activity that is to occur and passes control to the appropriate handler. As all transactions must conform to standard HL7 messaging formats, determining the transaction type is relatively straightforward. HL7 message formats are described in the Application Developer’s Guide document.

3) **PharmaNet Xfer**

This module gets control if the HL7 Msg Identifier module determines this transaction corresponds to a PharmaNet transaction. It formulates the call to the HNLLI library, makes the call and waits for the response.

4) **File Xfer Processor**

This module causes a file to be either sent to the remote server, or to be requested from the remote server. Further, this file may be either supplied from or to the client application over the socket or to a shared location on a file system.

5) **HL7 Xfer Processor**

This module causes a standard HL7 (i.e., not a PharmaNet nor File Transfer Request) transaction to occur.

6) **Client Config Module**

This module keeps track of all HNClient parameters defined during the HNSETUP process. In addition, it tracks changes to the information in remote facilities. This module does not permit changes to be made to the parameters set-up by the HNSETUP program. This module is also responsible for the facility password to be supplied, as it is this module which needs to decode the private key information.
7) Trace Facilities

This module is only active when HNClient has been started with the –t option. When active, this module
dumps diagnostic and trace information for each connection request made as well as total elapsed time,
transaction type and success or failure of the transaction. This module is structured in such a way as to
ensure that trace information from multiple threads do not overwrite each other or mix together (i.e., it
is threadsafe). The file being written to is always in the same directory as the HNClient executable and is
always named: HNCLIENT.TRC.

8) Error Response Facilities

This module is responsible for generating appropriate HL7 responses to various errors. This module is
not called for errors that are generated by remote applications. As far as the local system is concerned,
these types of responses are perfectly valid. This module is only called in cases where no connection
could be made, the format of the transaction was bad, network timeouts, or some other local
detectable error.
3.2 Set-up and Administration

Each targeted platform requires its own version of HNClient executable and run-time support files. These support files are generally available as part of the compiler suite you are using to generate the library. The only requirement for these files is that they be placed in the installation directory, a system directory or some other directory in the system’s search path.

Outside of the actual runnable code, there are two other configuration files which define the set-up, location, name, and other required information in order to communicate using HNSecure. These files, the Facility ID Profile and Client Profile, are described in the HNSecure: HNClient & HNServer Design Specifications document.

3.3 Startup

The HNClient service/interface exists as a separate process running on either the local machine or on another machine located on a secure local LAN. The interface is activated by an operator/user who must supply the password that enables access to the encrypted Private Key.

It is important to stress that if HNClient is being run on a separate machine from the calling application, the connection between the two machines must be secure. This is because communications between the client application and HNClient is over normal non-secure IP sockets. HNClient was intentionally designed to run separately from its client applications, effectively isolating them from encryption details. In some environments, it may be necessary to run HNClient on a separate physical machine from its client application(s), as a gateway. In these cases, it will be necessary to secure the connection between the two machines using physical security or some form of encryption.
3.4 Operation

3.4.1 Client Application Interaction

HNClient is a multi-threaded listener that spawns a thread for each active application connection.

Communicating with HNClient requires the client application perform the following steps:

1) Open a standard TCP/IP socket connection with the HNClient service. There is nothing special about this socket. No special set-up or other information needs to be provided to HNClient. Further, the client application may set the socket up to be in blocking or non-blocking mode.

2) Formulate a standard HNSecure HL7 message. HL7 Message Formats are described in the Application Developers Guide document. This may involve a standard HL7 transaction or a file transfer request.

3) Send the HL7 message over the opened socket. Note that no encryption is done at this point; HNClient handles all encryption before it passes the message along to the remote server. The HL7 message must be preceded by a header block, as described in the HNClient Communication Protocol section of this document.

4) Wait for a response on the same socket. Timeouts may occur at any point. If the socket dies, or the client waits too long, then it may be assumed that communication failed for some reason. In this case HNClient will close the socket with the waiting application indicating a timeout occurred.

5) Note that this may place the client application into a "quandary"; if the transaction sent to the remote server involved some sort of database update, the client will not know if the communications failed before the update occurred or after. It will have to perform a query to determine if the update succeeded.

6) An HL7 response is received over the socket in plain text. This may be in the form of actual data, an acknowledgement, or an error message. Note that no decryption is required, HNClient handles all decryption and verification of the data received.

7) The socket is automatically shut down by HNClient and should also be closed by the client application. No further communication is possible using this socket. To perform another transaction, all the above steps should be performed again.
3.4.2 HNClient Communication Protocol

Transactions between HNAPI and HNClient always involve standard HL7 messages. However, because of the difficult parsing requirements of the entire HL7 definition, a control block is prefixed to the HL7 transaction to inform both HNClient and HNAPI on return exactly how much data makes up the entire HL7 message.

This is necessary because in addition to standard HL7 messages, this library may append binary data (such as file contents) just past the end of the HL7 message. The length of the HL7 determines where the HL7 message ends and the start of the extra binary data begins.

The format of this protocol is as follows:

Table 1  HNClient communication protocol format

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDRCOMPID</td>
<td>char(2)</td>
<td>Component ID (see below).</td>
</tr>
<tr>
<td>HDRCOMPLENGTH</td>
<td>char(10)</td>
<td>Length of Data section. Right justified, zero filled. Maximum value 4 GB.</td>
</tr>
<tr>
<td>HDRCOMPDATA</td>
<td>char(var)</td>
<td>Actual data sent in this HN6 Component. Must be precisely the same length as specified in HDRCOMPLENGTH.</td>
</tr>
</tbody>
</table>
The HDRCOMPID field indicates what the data section contains.

Valid HDRCOMPIDs include:

<table>
<thead>
<tr>
<th>HDRCOMPID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>Contains binary file contents “in-stream”. The actual contents of the file are transmitted through the open socket. This may either be transmitted to or from HNClient.</td>
</tr>
<tr>
<td>FN</td>
<td>Specifies the name of a fully qualified file name available to both the local application and HNClient. This file specification must name a location in a shared file system. This segment may be either transmitted to or from HNClient.</td>
</tr>
<tr>
<td>HB</td>
<td>This is a Heart Beat segment. Because some transactions may take an arbitrarily long period of time to perform especially with large file transfers, this segment indicates activity is still occurring downstream from HNClient. This heart beat segment will be returned to the client application now and then and can be safely ignored by the client application except to reset its network inactivity timer back to zero. A Heart Beat segment always consists of a zero length data portion.</td>
</tr>
<tr>
<td>DT</td>
<td>These components hold data in the data section (normally HL7 messages). There is no restriction on the format of the data. A special property of this segment is that it must appear in every transaction. Because of this, we can require it to be the last segment transmitted to and from HNClient, thus, indicating to both ends that no more information will be coming.</td>
</tr>
</tbody>
</table>
3.5 Shutdown

The following events occur when HNClient is shut down:

If HNClient is shutdown because of a Ctrl-C or certain other software trappable events, a Normal Shutdown occurs. In this situation:

1) No further transaction requests are accepted from client applications.

2) Each pending transaction can complete. This may take some time if large file transfers are taking place.

3) A check is made in the temporary file area for transaction files that were not deleted for some reason. These files are deleted now.

If HNClient is shutdown because of a machine crash, the process died, or any other reason not caught in the normal situation, an Abnormal Shutdown occurs. In this situation:

1) All pending transactions are dropped immediately. Client applications are not given any reason for the transmission failure. Client applications will have no idea at what point their transactions were terminated.

2) All temporary files generated as a result of in-stream file transfers will remain. This implies a possible security risk; these files should be deleted manually or a new instance of HNClient should be started as soon as possible which will clean these files up.
4.0  HNLLI

4.1  Overview

HNLLI is designed for applications that do not wish to communicate over an open socket interface or for platforms that have unique multi-process/multi-thread capabilities.

HNLLI is a multithread safe library of routines which client applications link to through a direct interface DLL. The HNLLI library provides connectivity and data request capabilities to client applications. It accepts transactions (messages) from the client, routing them to the specified server application (local or remote). It provides all the functionality of the HNClient interface except at a much lower level. HNLLI is linked to the client application either statically or dynamically depending on the desires of the application writer and the capabilities of the target client platform.

The client application must perform several items that were taken care of automatically by HNClient. In fact, HNClient was written so that it communicates through the HNLLI library. The HNLLI library itself consists of a single ‘C’ function through which all requests, files, data, and responses flow to a destination server to be processed, stored and/or returned.

Each transaction in the HNLLI library sent to a server is:

- atomic (completely processed or reversed, i.e., no action left dangling);
- stateless (does not depend on the results of previous transactions);
- confirmed (not complete until a response or error code is returned); and
- synchronous (the client thread waits for a response from the server before proceeding).

This implies that no conversations between the client and the remote server take place. A request is sent to the server and a reply is received. Each request is separate and independent of all others which take place before it. This does not mean however, that the client application cannot add context to the contents of the transaction.
Normal interaction with HNLLI involves a significant amount of processing overhead on the part of the client application.

1) A valid TCP/IP stack must be initialized if not already available.

2) The facility’s private key must be obtained and decrypted.

3) The facility’s ID profile must be obtained.

4) Transmit, Receive and optional Trace buffers must be created.

5) Any information regarding the destination facility must be obtained. At a minimum, the facility ID and receiving application must be known.

6) The HNSecure HL7 message must be formulated.

7) The transaction must be placed in memory buffers along with the client private key, the remote facility ID, connection profile and the buffers.

8) When a response has been received, the function return code must be examined for success or failure.

9) Any updated information about the remote facility connection profile should be saved (cached).

10) The data from the remote server must be processed (if any was received).

11) The buffers should be deleted if they are not going to be used again.

4.2 Set-up

An equivalent set-up procedure to that specified in the HNSecure: HNClient & HNServer Design Specification document for HNClient needs to be followed to obtain all the information required to set-up communications with a remote server. Note that while HNClient set-up places all the configuration information into files, there is no requirement that this occurs for clients communicating through HNLLI. Clients may choose to store the information in a database, system registries, or through some alternate means.
4.3 Startup

The design of HNLLI provides a mechanism which does not require explicit start up or shut down by client applications, the client may make immediate use of the library. Any initialization which needs to be performed is done automatically by the library itself either when the library is loaded and unloaded by the operating system, or through automated checks in the library code itself.

There are several steps that must be performed to send a transaction or file transfer request to a remote server:

1) A valid TCP/IP stack must be initialized if not already available. In most environments, this normally occurs during system start up. In others, it may require intervention on the part of the application or user.

2) The facility’s private key must be obtained. The mechanism by which this key is obtained must protect the confidentiality of the private key. If this key is compromised, the confidentiality of HNSecure information is at risk.

3) The facility’s ID profile must be obtained. Usually this information is stored in a file, none of which needs to be considered private.

4) Transmit, Receive and optional Trace buffers must be created.

5) Any information regarding the destination facility must be obtained. At a minimum, the facility ID and domain must be known.

6) The HNSecure HL7 message must be formulated.

7) The transaction must be placed in memory buffers along with the private key, facility ID, connection profile and buffers. The thread making the request blocks until a response of some sort from the remote server is received or a network timeout occurs.

8) When a response has been received, the function return code must be examined for success or failure.

9) Any updated information about the remote facility connection profile should be saved (cached).

10) The data from the remote server must be processed (if any was received).

11) The buffers should be deleted if they are not going to be used again.
4.4 Operation

Operation and interaction with the HNLLI library is performed strictly through a ‘C’ style call interface that is either dynamically or statically linked into the client application. This method provides for maximum performance but also requires the greatest work on the part of the client application to prepare for successful communications.

Each process in an application may have any number of threads simultaneously sending transactions to the library. The library is capable of handling multiple requests from a single process, one request per thread.

4.4.3 HNLLI Interface

The HNLLI Interface has no user level interface, as it is strictly a code library. Instead it is controlled through calls to the HNLLI function. This interface allows client applications to submit transactions to the ministry HIE servers, such as sending HL7 transaction messages or performing file transfers either to or from the local machine.

4.4.3.1 HNID Structure

The Facility Identification Structure, and matching profile file, contains information which identifies the local network facility associated with this message or file transfer activity. All applications in this network-facility must have read access to this configuration file (or a copy of it). Note that this information does not need to be stored in an actual file, it may reside in a database, system registry, environment variables or any other source of information.

The HNID structure may be populated from any number of different sources. HNCClient populates it from a Facility Configuration file which is found in the same directory as the executable and has the same name as the instance when HNCClient is started with a .ID extension. See the HNSecure: HNCClient & HNServer Design Specifications document for a detailed description.

4.4.3.2 HNCP Structure

The Connection Profile structure contains information that identifies the server network connection to be used for this message or file transfer activity. A separate Connection Profile is required for each application server in each processing domain.

If the refreshed flag is set upon return, the application should store (cache) the new values for use when another connection with this server is required. This will ensure that the next connection request uses the most recent directory addressing and encryption information and avoids the overhead and time delay associated with refreshing this data for subsequent HNLLI calls.

See the HNSecure: HNCClient & HNServer Design Specifications document for a detailed description.
4.4.3.3  HNDATA/BC Structure

The HNDATA/BC structure contains the transaction and file related parameters required by HNLLI. Some of the fields within this structure are passed into HNLLI through this structure; others are supplied by HNLLI upon return.

See the *HNClient & HNServer Technical Specifications* document for a detailed description.

4.4.3.4  HNetCallBackObject

This structure provides the HNLLI library with information on how to provide intermediate call backs to the calling application. This allows the client to be assured that information is in fact being transferred during lengthy transactions and file transfers. Timeouts will still occur normally whether call backs are being performed.

In addition to this simple reassurance, this function also provides details on how far along in the current transfer HNLLI has gotten. This completion meter may be useful in estimating how much longer the transfer will take.

See the HNSecure: HNClient & HNServer Design Specifications document for a detailed description.

4.4.3.5  Tracing

The HNLLI normally writes trace messages to the buffer specified in the HNLLI function call. It should be noted that tracing facilities are optional. If no trace buffer is specified (NULL tracebuffer parameter) no tracing will occur. If a buffer is supplied, it is assumed to be at least 32,740 bytes long (32K - 28 bytes). The HNLLI library guarantees that no tracing message will be longer than that no matter what transaction is performed or what file is being transferred.
4.4.4 Transaction Layer Function Types

The Transaction Layer message formats are based on HL7 Version 2.2 standards with locally defined application data segment extensions as required by individual applications. Messages are not encrypted at this level, as the Lower Level Protocol ensures the secure delivery of messages on behalf of the Transaction Layer.

All transactions passed on to a server and received from the server are formatted according to the HL7 2.2 message protocol. While the HNLLI does not process these structures, it does pass them on verbatim to the specified remote server. This library must receive the transactions from the client application in their final form to be transmitted.

4.4.5 Connecting to HNGARD

There is a three-step process by which HNLLI connects to HNGARD:

1) Randomly selects one of the two HNGARD servers, uses it as the primary server and attempts a connection through it.

2) If that connection attempt fails, attempts a connection through the other HNGARD server.

3) If neither of these attempts succeeds, then a connection failure message is returned to the client application and the transaction fails. This is a significant issue for the client application because if HNLLI cannot connect to HNGARD, it cannot communicate with any remote site.

If the user is using Ministry approved compliant software, they should contact the PharmaNet Help Desk in Victoria at (250) 952-2867, Vancouver (604) 682-7120 or Rest of BC 1-800-554-0225. If the user is developing an application, they should contact Hlth.CISSupport@gov.bc.ca for assistance.
4.5 Shutdown

As the library is integrally linked into the client application, it is up to the client application to determine how shutdown is to occur. The client must decide the fate of pending transactions and file transfers.

There are no specific requirements of HNLLI that must be performed prior to shut down as no state information or set-up conditions are recorded by this library.
5.0 Security Services

The HNClient/HNLLI libraries provide secure transmission from a client to a remote server. This implies that any data sent from the library to the remote server has the following properties:

- The data is received by the remote facility specified;
- The data cannot be read by other parties; and,
- The data cannot be modified by other parties without the destination facility detecting that a change occurred.

Data received by the remote server also has similar properties:

- The data received from a remote facility actually came from the facility indicated;
- The data cannot be read by other parties; and,
- The data cannot be modified en route by other parties without it being detected.

The security model described here details the steps performed in order to deliver data securely. Knowledge of the information in this document does not compromise the security of this model.
5.1 Public/Private Key Encryption

This method of encryption is more secure than DES, however that extra security comes with a heavy processing penalty. For this reason, this method is used to encrypt the DES key that will be used for encrypting the rest of the transaction. Because a common DES key must be used at both ends of the transmission, the server must be provided with this DES key.

For the server to receive the key, the client HNAPI encrypts the DES key to be used with the server’s public key (found in the HNCP structure). The server then decodes the key using its own private key. If the decoding works, the client can be assured it is being processed by the correct server.

A message digest is computed on the data to be sent and encrypted using the client’s private key, creating a digital signature. If the signature can be verified, using the sending facility’s public key at the server facility, the server can be assured that the message came from the correct client.

From that point on, any data transmitted to the remote server will be encrypted using the DES key.

When the remote server responds, it formulates the response and encrypts using the same DES key as the transaction. It also signs the response in the same manner as the client signed the transaction. The client then computes the response message digest and compares it to the signature message digest. If everything matches, the data and response are considered to have been transferred securely.

5.2 DES Key Encryption

This method of encryption, while not as secure as Public/Private Key encryption is much faster and is very secure as the DES key is used only once (a new DES key is generated for each transaction). For this reason, all sensitive data that must be passed to and from the server will be encrypted using the DES algorithm. HNSecure uses the triple-DES (3DES) variant of the DES algorithm.

The operation of the DES key generation, encryption, and decoding is provided through third party software (i.e., the Certicom toolkit).
6.0 Sample Applications

Two sample applications are provided which demonstrate the capabilities of HNClient/HNLLI and how they interact with the server. These samples also serve as a template to application developers who must use this library.

These applications are very simple with simple user interfaces. They are not designed to be comprehensive but provide a proof of concept and example of use. Please note that these applications are not HNSecure compliant in that they do not perform any user authentication.

6.1 Batch HL7

This program is a command line application that takes as input a file containing one or more HL7 transactions and transmits them to one or more HIE servers. The purpose of this application is two-fold: to exercise the Client libraries with respect to HNSecure HL7 messages; and to exercise server applications in how they handle HL7 messages sent to them.

6.2 HNTIME

This very simple program demonstrates the minimal form a HNSecure application may take.
7.0 HN6 Lower Level Protocol and Message Formats

The Lower Level Protocols (HN6) roughly corresponds to OSI level 6. This level is responsible for handling socket connections, including the sending and receiving of messages over the socket. This is also the level at which security is implemented. The Security Model and Services are described more fully in a previous section of this document.

7.1 Communications Protocol

This module of the library provides all of the communications with remote servers. This is done over a TCP/IP connection.

It is anticipated that all platforms to which this library will be ported will have some form of the Berkeley Sockets interface for communicating over the Internet available for use. This interface was chosen because Berkeley Sockets is the de-facto standard for API’s using the Internet.

The initial version of the client side of the library is implemented on Windows using the WinSock library as well as IBM AIX systems. This library implements a Berkeley Sockets interface as well as their own extensions. All attempts will be made to avoid use of the extensions and deal with pure Berkeley Sockets function calls.

The following diagram illustrates the full life cycle of a client socket connection to the gateway.

Figure 4 HNLLI and HNGate connection sequence diagram
The processing of a transaction may, require the opening a socket, or some other kind of connection to a final server application but this will be totally transparent to the client application. The client-gateway socket connection will be maintained throughout the entire life time of the transaction as indicated in the diagram. Once the transaction has been completed the socket will be closed. Future transactions will require opening new sockets with the server.

Detailed information regarding the HN6 Communications Protocol can be read in Section 4.7 Lower Level Protocol (HN6) of the HNSecure Technical Specifications document.
8.0 HNSETUP

8.1 Overview

This section of the document describes how HNSETUP goes about its business, including the format of the configuration files it produces. For more information on HNSETUP refer to the Client/Server Design Specifications document.

The HNSETUP program is a straightforward step by step program which initializes and maintains the configuration files which particular instances of HNClient depend on.

Once a configuration has been set up, updates to this configuration may be performed as needed. This is useful to manually change HNGARD settings (e.g., passwords, listening port, open socket queue size).

The final product delivered by HNSETUP is a single file (the .ID file) holding the facility’s identification and configuration parameters as well as an encrypted private key for the facility. The private key is always protected by a password scheme.

The following diagram illustrates the functional blocks of HNSETUP:

![HNSETUP functional blocks](image)

*Figure 5 HNSETUP functional blocks*

This diagram demonstrates that HNSETUP is a client application to the HNLLI library. This is because HNSETUP must communicate with HNGARD to complete set-up and register the new facility and/or update the facility’s public key.
8.2 Set-up and Administration

The only requirement for set-up and administration of the HNSETUP program is that it be placed in the same directory as the HNClient executable file.

When this program is run, it looks for and places all configuration files into this directory. HNClient also expects these configuration files to exist in this directory as well.

8.3 Startup

HNSETUP is designed to be a portable application and as such runs with a text only interface. The HNSecure Functional Specifications document describes how to start and use this program.

HNSETUP performs the following tasks at start up:

1) Ensures no instance of HNSETUP or HNClient is currently running. If one or the other is found, an error message is produced, and the program terminates.

2) It checks for existing configuration files. If none are found, it performs a new installation procedure. If at least one of the configuration files are found, it performs an interactive menu procedure.

If HNSETUP detects no configuration files, it will automatically begin a new installation. If there are configuration files present, the user is presented with a menu of choices as to what to do next. The Functional Specification also enumerates all these choices and requirements of the user.
8.4 Operation

Normal operation of HNSETUP is either as an automated installation process for new installations, or through a menu presented to the user. It is important to remember that the facility can stand on its own and be used without the use of HNClient or HNServer, but HNClient and HNServer must use the facility information to which it is bound.

There are four configuration files produced by HNSETUP:

1) Facility profile/configuration file
2) Affects the security and communications of the facility.
3) Client profile/configuration file
4) This contains parameters that affect HNClient only.
5) Server profile/configuration file
6) This contains parameters that affect HNServer only.
7) Transaction profile/configuration file
8) This contains parameters that affect HNServer only.

See the Client/Server Design Specifications document for more details on the configuration files.
8.5 Public/Private Key Generation

When HNSETUP wishes to register with HNGARD, it uses the HNLLI library to perform the remote registration. Part of the information it must pass to HNGARD is the public key it wishes to use for secure communications. The steps it takes to do this are:

1) Ask the user for the Facility Registration Number.

2) Ask the user for a facility password and confirm it. This will become the password for HNClient.

3) Generate a Private/Public Key pair.

4) Send an HN6 Registration Message to one of the HNGARD servers. If this fails, try the other HNGARD server. Refer to the Application Developer’s Guide document for information regarding this message. The Facility Registration Number is used to encrypt this transaction, effectively ‘signing’ it.

5) If successfully registered, encrypt and write the private key to the HNClient.ID file using the supplied facility password. Inform the user of successful installation.

6) If the response was unsuccessful, inform the user of the problem (e.g., domain error, cryptographic error, unauthorized). The private key is still encrypted and written to the HNCLIENT.ID file, so that the registration can be retried at a later time.

7) If a Network Timeout occurred, then either the registration never occurred, or it occurred and failed, or it occurred and succeeded. In any case, write out the encrypted Private Key to the HNCLIENT.ID and inform the user of the problem. If the registration was not completed or it completed in error, HNClient will not work.

It is important to note that these registrations for HNClient and HNServer are separate registrations, and to register both requires that the user do two separate registrations. The client and the server will both have their own separate public keys registered with HNGARD.
8.6 Shutdown

There are no special considerations when shutting down HNSETUP. If the program crashes or otherwise terminates improperly, HNCclient will fail to start or operate properly, prompting for HNSETUP to be run again.

If HNSETUP is run again after a crash, it will be able to detect how far it got and continue from there based on the state of the configuration files.
9.0 HNSERVER

9.1 HNSERVER Interface

The server interface provides a mechanism for a facility to listen for incoming messages, have them processed, and return the response back to the originating client.

The server interface is an optional component of the overall package. It is not necessary to set up the HNServer portion to use the Client Interface as described above.

The server interface employs the same security model as employed by the client portion of this package in that it authenticates facilities rather than the applications or the users employing them.

There is a single application interface for application developers to work with. This IP based interface is identical in format to the interface between HNAPI and HNClient.

There are two functional parts to the server interface:

| 1. HNServer Service | HNServer waits for HL7 transactions to be sent to it from any client facility operating on the HNSecure network.  
Each transaction is then examined to determine the transaction type, and based on this, it routes the transaction to the appropriate server application for processing.  
The message is routed to the server application using a simple protocol identical to the protocol used to route messages from HNAPI to HNClient. |
|----------------------|------------------------------------------------------------------------------------------------|
| 2. Server Applications | These applications wait for HL7 messages to be passed to them from HNServer.  
The protocol used to transmit to them is identical to the protocol used to transmit from HNAPI to HNClient.  
Once a transaction has been received, it is processed, and the response is returned over the same channel.  
The response will ultimately be received by the client that originated the HL7 message.  
Each server application processes distinct HL7 transactions (though each may service any number of different transactions).  
In the context of this package, the HNServer portion is supplied with the installation set, however each application server must be separately installed and configured on a site by site basis.  
A sample server application is supplied for developers to provide a template of how to build their own server applications. |
9.2 HNSERVER (Service) Overview

The HNServer is an optional product that need not be set up for a site. It is provided for those organizations that have specific processing needs not provided through standard ministry HIE systems.

The HNServer Service provides a simple, non-direct interface for client applications wishing specific proprietary message processing for various types of HL7 messages. The only access through HNServer from an originating client is through the HNSecure Client software. The only access to the back-end server applications is through standard sockets. It is important to note that the connection to backend servers is not a secure communications channel unlike the connection from the client application to HNServer which is. The HL7XFER library has been written to facilitate the communication between HNServer and the message processing application.

All communications from HNClient with HNServer are identical to communicating with HNGATE so no special considerations need to be considered when sending HL7 messages to an HNServer except appropriate formatting of an HL7 message.

Figure 6 HNClient to HNServer communication example
9.3 Server Application (Service) Overview

An installation may have any number of server applications running, each processing their own distinct set of HL7 messages. To be useful, a site should run at least one server application.

Each server application can process any number of different HL7 messages. The messages processed by individual application servers are defined by the Receiving Application field of the HL7 MSH segment.

An application server listens on a known IP address and port number for connections from HNServer. Any HL7 messages it receives, it processes, in turn producing a resulting HL7 message which is then returned to the HNServer. The basic steps are detailed below:

9.3.1 Basic Message Processing

To accept and process incoming HL7 messages, the following steps are performed:

1) A socket is set up to listen for incoming messages.

2) A connection is detected.

3) The HL7 message and optional file attachment are sent to the application server from HNServer using the same protocol as that between HNAPI and HNClient.

4) The HL7 message is handed off to the actual routine or process to handle the message. In the meantime, heartbeat messages are returned to HNServer to keep the connection alive.

5) A response HL7 message is formulated.

6) The response message and optional file attachment is returned to HNServer.

7) The socket for this connection is closed. Multiple messages are performed by looping back to step 2.

The interface between HNServer and the application server is not encrypted requiring this connection be physically secure.

While not strictly necessary, most implementations would set up the application server to support multi-processing/multi-threading. This would permit better support of concurrent connections and message processing.
9.4 HNSERVER Function

The following diagram shows the functional dependencies of the HNServer service:

![Diagram of HNServer functional dependencies]

Figure 7 HNServer service functional dependencies

As can be seen in this diagram, there are seven modules central to the operation of the HNServer service:

1) HL7 Validation

This module accepts the incoming HL7 message from the remote client and checks the MSH for completeness and consistency. It extracts various fields from the MSH and compares them against values it expects to find or not find. If it does not pass, the message is rejected, and an error response is returned to the remote client facility.

This module also ensures that for operator commands, the incoming message was received by a facility which is identified as an operator facility.

This module does not concern itself with anything past the MSH segment nor does it concern itself with fields within the MSH for which it does not check.
2) HL7 Message Router

This module examines the Receiving Application field for the server application processor it needs to deliver the HL7 message to. This is accomplished by looking up the Receiving Application field in the message routing table (part of the Server Configuration files).

There are two possibilities for the routing:

- an internally processed message in which case this module passes it to the Internal HL7 Application Server; and,
- an externally processed message in which case the IP address and Port number of the listening application server is retrieved and the HL7 message, optional file and IP information is passed on to HL7XFER Client module.

This module passes received HL7 messages from a remote facility on to a listening HL7 Application Server for processing and eventual response. It uses the HL7XFER protocol to transmit and receive the HL7 messages and optionally attached files.

3) Internal HL7 Application Server

This module will process all HL7 message destined directly for the HNServer service. Typical HL7 messages of this type would be date time requests and operator commands. This module generates a complete HL7 response and returns through the call chain for eventual return back to the originating client.

4) HL7XFER Client

This module is called when a message is destined for an external application server. This server may exist on the same machine or a remote machine, the parameters of which are passed in from the HL7 Message Router.

This module will forward the HL7 message and optionally attached file to the specified application server and wait for a response. When a response is received, it will return the HL7 response and any optionally attached file back through the chain of modules and eventually back to the originating facility.

This module uses the HL7XFER protocol which is described in full later in this document.
5) Server Config Module

This module keeps track of all HNServer parameters defined during the HNSETUP process. It also retrieves the transaction routing table which identifies the application server and its contact information for each HL7 transaction known by this HNServer instance. Finally, it tracks changes to the information regarding remote facilities.

This module does not permit changes to be made to the parameters set-up by the HNSETUP program or to the transaction table.

6) Trace Facilities

This module is only active when HNServer has been started with the –t option. When active, this module dumps diagnostic and trace information for each connection request made as well as total elapsed time, transaction type and success or failure of the transaction.

This module is structured in such a way as to ensure that trace information from multiple threads do not overwrite each other or mix together (i.e., it is thread safe).

The files being written to are always in the same directory as the HNServer executable.

7) Error Response Facilities

This module is responsible for generating appropriate HL7 responses to various errors. This module is not called for errors that are generated by remote applications. As far as the local system is concerned, these types of responses are perfectly valid. This module is only called in cases where no connection could be made, the format of the transaction was bad, network timeouts, or some other local detectable error.
9.5 Set-up

9.5.1 Overview

The HNServer Service provides a standard TCP/IP socket interface to secure communications with local or remote HNSecure clients and locally available application servers. HNClient communicates with the HNServer by opening sockets and sending a secure transaction using the HN6 secure communications protocol.

Each targeted platform requires its own version of HNServer executable and associated run-time support files. However, these support files are generally available as part of the compiler suite you are using to generate the program. The only requirement for these files is that they be placed in the installation directory, a system directory or some other directory in the system’s search path and that they be multi-thread safe.

Outside of the actual runnable code, there are three other configuration files which define the set-up, location, name, and other required information to communicate using HNSecure. These files, the Server Configuration File and Transaction Routing File, are described in the HNServer section of the HNSecure: HNClient & HNServer Design Specifications document and are managed by HNSETUP.

These configuration files must be placed in the same directory as the executable for HNServer.

It is important to note that the communication between an HNServer machine and a LAN based application server does not incorporate any security whatsoever. It is up to the network administration to ensure that data transmitted from HNServer to the applications servers can be done with minimal risk of compromised data. It is also for this reason that it is essential that the HNServer Service be on the same LAN as the application servers.

If the security cannot be ensured, or HNServer cannot be placed in the same LAN as the application servers, then HNServer must be installed on the same machines as the application servers.

The HNServer Service must be set up as a separate process on either the local machine on which the application servers are to run, or on another machine on the local LAN to which the machine has access. In either case, communication between HNServer and applications is performed using standard TCP/IP sockets.

A program named HNSETUP.EXE is used to set up a server node and register it with HNGARD. Use of this program allows an initial configuration as well as periodic updates as the local environment changes or password changes are desired. After successful initialization, all remote HNSecure clients will have secure access to registered HNServer services through a standard TCP/IP socket.
9.5.2 Requirements

Each instance of the HNServer Service represents a single network node with its own network facility ID. If a copy of HNServer is to be installed on individual machines, each machine will be assigned a unique network facility ID along with the unique encryption keys for message security.

Prior to set-up of HNServer, an on-line registration form (https://www.health.gov.bc.ca/exforms/das/hnsecure.html) must be completed, which is automatically sent to Conformance and Integration Services (CIS). CIS will validate the application and return the parameters required by the set-up program. These parameters are necessary prior to setting up HNServer and must be kept on hand if reconfigurations or reinstallations are necessary. These parameters must also be kept confidential as they identify a facility to HNGARD.
There are two configuration files that help to identify an instance of HNServer. The first file, the ID profile file, is called HNSERVER.CFG and contains the following information:

- instance name (always HNServer);
- facility ID;
- processing domain;
- IP addresses and port numbers of the two HNGARD servers this HNServer will be using. This also includes the DNS names of these two HNGard servers to use in case the IP addresses must be refreshed;
- client network timeout period;
- application server timeout period;
- the public and private keys for the facility (the private key is encrypted with the facility password);
- public key of the HNGARD servers used by this client;
- IP address and port number to listen on for connections from HNClient;
- IP address and port number to listen on for operator commands;
- maximum number of pending client connections it will allow;
- logging level flag indicating the amount of detail to dump to an external log file;
- directory to log transaction requests and responses to;
- name of the transaction profile file to get information about known transactions; and
- directory to store received files from HNClient.
The second configuration file details how HNServer should process individual transactions it receives from HNClient. This file is named as specified in the HNSERVER.CFG file and contains the following information:

- transaction name (as it appears in the receiving application field of the MSH);
- transaction type (Operator or Normal);
- server type (In process, TCP/IP connection, or Forward to other gate);
- IP address of listening application server which handles this message;
- IP Port of listening application server;
- Forwarding facility ID; and,
- Enabled flag.

These tables are detailed further in the HNSecure: HNClient & HNServer Design Specifications document Configuration Files section.

### 9.5.3 Other Set-up Notes

There are set-up requirements that may need to be performed by the administrator and are considered separate from the set-up of HNServer. These may include:

- Ensuring that a valid TCP/IP stack can be created with an appropriate transport mechanism.
- Ensuring LAN security meets the current requirements as per the Privacy and Security Conformance Standards.
9.6 Startup

To provide services to client facilities, the HNServer program must be started as a separate process from all other HL7 application servers. The HL7 servers may be started and stopped as desired as they are dynamically connected to as each HL7 message is directed to them. The mechanism for starting and stopping the application servers is out of scope for this document.

The HNServer service is started from the command line with an option to enable diagnostic information dumps as HNServer runs and processes connections from client facilities using HNSecure.

Once started, the operator is queried for a Facility Password which unlocks the Facility Private Key and enables communications with HNSecure-compliant client facilities and the Ministry of Health HNGARD.

It is important to stress that if HNServer is being run on a separate machine from the HL7 Application Servers, the connection between the two machines must be secure. This is because communications between the application server and HNServer is over normal non-secure IP sockets using the HL7XFER protocol. HNServer was intentionally designed to run separately from its HL7 Application Servers, effectively isolating them from encryption details.

The following steps occur when HNServer is started.

1. The service is started with a command line with the following form:

   
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HNSERVER [-t]</td>
</tr>
<tr>
<td>-t</td>
</tr>
</tbody>
</table>

   -t
   Enables an optional trace log to be created. Generates a file named:
   HNSERVER.TRC
   An optional flag that causes HNServer to generate a trace file named
   HNSERVER.TRC. If this file exists, it will be overwritten. This tracing information is
   in addition to other logging facilities generated by HNServer.
   The trace option should not be specified for regular production use, as it can
   generate an enormous volume of trace data.
The steps required for the operator to start HNServer are specified fully in *HNClient & HNServer Technical Specifications* document.

1) All configuration files are read in. The HNServer Configuration File and the HNServer Facility ID file must be placed in the same directory as the executable program. The configuration file also has a pointer to the transaction profile file which specifies which transactions are supported and how to route them. This file is also read in.

2) A temporary file area is set up for this instance. In this area, files generated as part of the in-stream file transfers will be stored for this instance. This file area must be distinct from all other instances of HNServer currently running.

3) A check is made of the temporary file area for this instance. If any files exist in this area for any reason (perhaps an earlier abnormal termination), these files are deleted.

4) A password is requested from the user. This password is immediately scrambled with a call to `hnlli_preparePassword` and this scrambled password is checked against the password used to encrypt the private key information.

5) HNServer starts two threads that listen for socket connections on the Transaction Port Number and Operator Port Number specified in the HNSERVER.CFG file for this instance.

HNServer does not attempt to contact the application servers specified in the transaction profile to ensure they are available.

### 9.6.4 Basic Client Connection Steps

The connection requirements for the client software (handled automatically through HNLLI) are listed below:

1) The client retrieves the connection information from HNGARD (if not already cached locally).

2) Using the connection information, an IP socket is opened with the desired HNServer.

3) The HN6 protocol is used to send the message to the desired HNServer.

4) The client application waits while HNServer sends the message to the appropriate server application for processing.

5) HNServer sends the response (or generates an error response) back to the originating client using the HN6 protocol.

6) The client receives the response and closes the connection with HNServer.
9.6.5 Basic Server Application Connection Steps

Once HNServer has received an HL7 message from a client application using the HN6 protocol, the message must be routed through to a server application for actual processing. To do this, HNServer performs the following steps:

1) HNServer scans the HL7 MSH segment for the Receiving Application field.

2) HNServer looks up in a table containing connection information which listening application server will process the message.

3) An IP socket with the listening server application is opened.

4) The HL7 message is transmitted (in the clear) to the server application for processing. The protocol describing this transmission is discussed in the HNClient Communication Protocol section of this document.

5) HNServer waits for the reply to come back from the server application.

6) The response is received (in the clear) or an error response is generated by HNServer.

7) The connection with the server application is closed.

8) The resulting HL7 message is returned to the originating client using the HN6 protocol.

While not strictly necessary, most implementations of HNServer will be set up to support multiprocessing/multi-threading. Each connection from an originating client causes a new thread to be launched within HNServer and processed independently of all other concurrent connections.

The connection between HNServer and the application servers employs no software security mechanisms unlike the connection between HNClient and HNServer. Because of this, these connections must be protected in some other secure manner.
9.7 Operation

9.7.1 Server Application Interaction

HNServer is a multi-threaded listener that spawns a thread for each active application connection. Each of these threads communicates with a back-end application server for final HL7 message processing. For this reason, these application servers should be multi-threaded as well. Alternatively, HNServer or the application server will have to guard against multiple connections to application servers.

Communicating with the application server requires HNServer to perform the following steps:

1) HNServer detects a connection from a client facility using the HN6 Lower Level Protocol (see the section 4.7 in the HNClient & HNServer Technical Specifications document for a description of this protocol).

2) HNServer receives the HL7 transaction and parses the MSH segment for transaction routing information. Specifically, it looks up the receiving application name from the MSH segment.

3) The receiving application name is used to consult the HNSERVER.TXN file to determine which application server to route the HL7 message (and optional file) to.

4) HNServer opens a standard TCP/IP socket connection with the application server. There is nothing special about this socket. The IP address and port number is retrieved from the transaction profile. No special set-up or other information needs to be provided to the application server. Further, the application server may set the socket up in blocking or non-blocking mode.

5) Send the HL7 message over the opened socket. Note that no encryption is done at this point, HNServer handled all encryption as it received the message but the connection to the server application is in the clear. The format of this protocol is described in the HNClient Communication Protocol section of this document.

6) Wait for a response on the same socket. Timeouts may occur at any point. If the socket dies or the client waits too long, then it may be assumed that communication failed for some reason. In this case HNServer will close the socket with the waiting application indicating a timeout occurred.

7) Note that this may place the client application into somewhat of a quandary; if the transaction sent to the remote server involved some sort of database update, the client will not know if the communications failed before the update occurred or after. It will have to perform some sort of query to determine if the update succeeded.

8) An HL7 response is received over the socket in plain text. This may be in the form of actual data, an acknowledgment, or an error message. Note that no decryption is required; HNServer will handle all security as it returns the response back to the client application.
9) The socket is automatically shutdown by HNServer and should also be closed by the client facility. No further communication is possible using this socket. To perform another transaction, all the above steps should be performed again.

9.7.2 HNSERVER Communication Protocol

Transactions between HNServer and server applications always involve standard HL7 messages. However, because of the difficult parsing requirements of the entire HL7 definition, a control block is prefixed to the HL7 transaction to inform both the server applications and HNServer on return exactly how much data makes up the entire HL7 message.
9.8  HNSERVER HL7 Transaction Routing

9.8.1  File Attachments to HL7 Transactions

The HNSecure message protocol allows files to be attached to HL7 transactions. The ‘message type’ field of the transaction’s MSH segment indicates if the transaction is ‘getting’ or ‘putting’ a file (‘ZFG’ for ‘gets’, ‘ZFP’ for ‘puts’) to/from the server, while the ‘continuation pointer’ field specifies the ‘network filename’ for the file to be transferred. The ‘network filename’ does not necessarily correspond to a real file name on any physical file system; it is merely a label for HL7 transactions to use with file attachments. Application developers can use the ‘network filename’ as they see fit.

When HNServer receives a transaction with a file attachment (a ‘put’ transaction), it saves the file to a unique random name in the directory specified by the external storage directory identified in the HNSERVER.CFG file. This unique random name is appended to the HL7 transaction (immediately following the double carriage-return terminator) for forwarding to the target server application. The server application is free to copy/move/delete the file as it sees fit.

When HNServer receives a file ‘get’ transaction, it expects the HL7 response from the server application to contain a filename immediately following the double carriage-return terminator. HNServer removes the filename from the HL7 response and forwards the contents of the file specified to the client in an HN6 BF block. If the file does not exist (or no filename was given), an empty BF block (i.e., zero length file) is returned.
9.9 Shutdown

HNServer can be shutdown by pressing Ctrl-C when the server process has the focus or when an HL7 operator message is sent to it to shutdown. Alternatively, HNServer can also be sent the operator transaction HNETSHUTDOWN (described in Section 7.4.2 of the HNClient & HNServer Technical Specifications). The Ctrl-C mechanism works identically to the HNETSHUTDOWN transaction.

As HNClient shuts down, no new connections will be allowed, allowing HNServer to close all outstanding communications links.

There are two modes of shutdown:

<table>
<thead>
<tr>
<th>1. Normal</th>
<th>When this occurs, all pending transactions can complete or timeout. Lengthy transactions or file transfers may need to be terminated prior to completion. This method of shutdown is activated using normal shutdown procedures such as Ctrl-C (for platforms that support this) within the client’s window or some other normal shutdown method such as the operator transaction HNETSHUTDOWN is sent.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Immediate</td>
<td>In this event, all pending and ongoing transactions are terminated immediately. No messages are generated, the socket is merely closed. The client will notice this as a loss of communications and need to react appropriately. This method occurs in the event of a machine failure, drive crash, power loss or some other catastrophic failure.</td>
</tr>
</tbody>
</table>
10.0 HL7XFER

10.1 Overview

This module handles all unsecured communications between the client application and/or HNAPI with HNCIClient and all communications between HNServer and the server application(s) attached to that server.

This module implements a simple yet flexible protocol that will send an HL7 message and an optional file to the other end of the link and then waits for a response. There are two halves to any communications link: the client or originator, and the server or listener.

The listener in turns waits for a connection, receives the message and optional file and gives these to the service waiting for this data. This module then waits for a response data set to be returned to the originator. While it is waiting, “keep-alive” notifications are sent to the connection originator to let it know that the connection is still alive and not to timeout.

An application wishing to transfer an HL7 message and optional file should use this library to perform the communications. Those applications that cannot link this library into their applications directly must implement the protocol described in this chapter as the listener side will be expecting it.

One important note about communicating using this mechanism is that all data is transmitted in plain text. This implies that the LAN must be physically secured from outside eavesdropping.
10.2 Communicating with an HL7XFER Listener

When originating a connection with a listener, the following steps are performed by the connection originator:

1) The IP address of the listener is resolved by the originator application. This occurs outside of the HL7XFER modules.

2) For HNAPI communicating with HNClient, the IP address is determined by a lookup into the \ETC\HOSTS file searching for an entry named HNClient. If no entry can be found, an IP address of 127.0.0.1 is assumed.

3) For HNServer communicating with an HL7 server application, the appropriate entry in the transaction routing table is consulted and the IP address is extracted from that.

4) The IP port number of the listener is resolved by the originator application. This also occurs outside of the HL7XFER modules.

5) For HNAPI communicating with HNClient, the port number is determined by a lookup into the \ETC\SERVICES file for an entry named HNClient. If no entry can be found a port number of 19430 is assumed.

6) For HNServer communicating with an HL7 server application, the appropriate entry in the transaction routing table is consulted and the Port number is extracted from that.

7) A standard IP socket is opened with the listener using the derived IP address and port number. This occurs outside of the HL7XFER module.

8) If a connection is made and a file or file name is to be transmitted to the listener, it is sent over the socket with either a BF or FN segment identifier (described in Section 3.4.2 of this document). This is handled by the HL7XFER library.

9) If a connection is made in step 3, the HL7 message is transmitted to the listener using a DT segment (described below). This is handled by the HL7XFER library.

10) A response is waited for from the listener. Either it will receive a proper response, or a network timeout will occur. While waiting, any number of HB segments may be received from the listener. These segments indicate that valid activity is still occurring on the listener side and not to timeout the connection. On receipt of these segments, the HL7XFER library resets its network inactivity timer back to zero and continues to wait.

11) If a file is to be returned from the listener as part of the response, a BF or FN segment is received. The HL7XFER library will store the file and record the name it was saved under.
12) The data segment containing the HL7 response message is received in a DT segment by the HL7XFER library.

13) The HL7 message and the file name containing the optional file is returned to the client application.

14) The socket is closed, and any cleanup is performed by the client application. This occurs outside of the HL7XFER library.
10.3 Operation

The HL7XFER library provides an automated means of sending HL7 messages between two machines. It does not interpret the contents of the HL7 in any way, so in fact, the data being transmitted could be garbage, but this will not affect this library in any way.

The library works from two points of view: that from the client or the originator, and from the listener or server point of view.

The interface from the client is very simple consisting of a single function call that will handle all communications with the listener side. Every message to be sent and responded to corresponds to a single call to this library.

The server-side interface is somewhat more complicated in that the process must, by necessity, be broken up into the discrete steps of creating the listener, waiting, receiving, responding, and closing the listener as other tasks need to take place between each of the steps such as HNServer forwarding the HL7 off to the application server for processing.

When errors occur, they typically cause the transaction to terminate. In the case of errors, this routine will only return an error code; specifically, it will not generate an HL7 error response. That is the responsibility of the calling routines.