FAA-HDBK-009-B
November 2012

FEDERAL AVIATION ADMINISTRATION HANDBOOK

Common Message Handling Protocol (CMHP)

This handbook is for guidance only. Do not cite this document as a requirement.

Foreword

1. This handbook is approved for use by all Departments of the Federal Aviation Administration (FAA).
2. This handbook documents the Common Message Handling Protocol (CMHP), which is an application-level protocol that runs over Transmission Control Protocol/Internet Protocol (TCP/IP).
3. This handbook is a reference document.
4. Address comments, suggestions, or questions regarding this handbook to:

   Enterprise Product Support Program Management Office (EPS PMO)
   FAA William J. Hughes Technical Center
   Atlantic City International Airport
   Atlantic City, NJ 08405

   e–mail: 9-ACT-FTI-PST@faa.gov

Table of Contents

List of Figures

List of Tables

1 SCOPE

1.1 Scope

This handbook provides the definitive detailed description of the Common Message Handling Protocol (CMHP) that should be used to support the software development and implementation of the protocol. In addition, it should be used as a reference document in the development of the Interface Requirement Document (IRD) and Interface Control Document (ICD) that will be developed by systems using CMHP.

CMHP is an application-level protocol developed by the FAA. It is designed for TCP/IP users. TCP/IP is a fire-and-forget protocol with no message receipt acknowledgment and little socket management. CMHP adds functionality (at the application level) that provides message-delivery assurance, regardless of the number of sockets that are traversed between end-systems.

The EPS PMO can provide CMHP source-code libraries under an "as-is" agreement to any user system that wants that to develop CMHP-based applications. The EPS PMO also provides potential users with the associated IRD and ICD to connect to the NADIN MSN, WMSCR, and ADAS.

This handbook describes how CMHP is implemented. It also provides the conformance test plan that the EPS PMO arranges to conduct with new user systems that want to connect to WMSCR, ADAS, or NADIN and who want to use CMHP features.

The scope of this handbook is restricted to the latest version of CMHP (v1.3), its predecessors CMHP v1.2 and CMHP v1.1, and obsoletes CMHP v1.0. The scope is further restricted to CMHP use over Transmission Control Protocol/Internet Protocol (TCP/IP).

Note: This handbook discusses the base functionality defined as part of CMHP v1.1. Any new functionality defined by CMHP v1.2 and CMHP v1.3 is called out explicitly.

This handbook provides:

- An overview of CMHP
- A description of the functions CMHP provides
- A technical description of CMHP message elements
- The CMHP conformance test plan, which includes an overview of the test cases and a strategy to execute them.

This handbook is for guidance only and cannot be cited as a requirement.

1.2 Change Record

Vertical lines in the margins (change bars) identify new or revised text added since the previous baseline.

Description Of Change	Revised Date	Version Number
Initial Baseline	May 2011	-
Added guidance regarding Poll and Final flags (Sections 5.2.10.1 and 5.2.10.3) Added further guidance on the use of status codes (Sections 4.3.2.2) Added further information on the validation of Major and Minor version fields and Source Location Identification field (Sections 4.3.6.3) Added new Section 4.3.8 Added clarification on user-defined Stop Service Notification messages (Section 5.3.4)	April 2012	A
Added three new conformance tests: R6-20, A2-17, and S2-08; modified D3-02 to accommodate an additional test scenario.	November 2012	B

2 APPLICABLE DOCUMENTS

The documents listed below are not necessarily all of the documents referenced herein, but are those needed to understand the information provided by this handbook.

2.1 FAA Documents

This section is not applicable to this handbook.

2.2 Non-FAA Government Documents

This section is not applicable to this handbook.

2.3 Non-Government Documents

The following documents form a part of this document to the extent specified herein.

a. IETF RFC 793 Transmission Control Protocol, September 1981
(http://www.ietf.org/rfc/rfc793.txt)

b. IEEE 802.3 Local Area Network (LAN) Protocols
(http://standards.ieee.org/getieee802/802.3.html)

3 DEFINITIONS

3.1 Terms

Application	The CMHP application is the software that performs the functionality described in this handbook. Application refers to the client and the server applications.
Far-end application	When two systems are communicating, the far-end application is the application on the remote system.
Local application	When two systems are communicating, the local application is the application at the near end.
RFS	Request For Service. This is a process within several FAA organizations that covers the authorization and connectivity of users to access FAA systems for application-level specific services.

3.2 Abbreviations and Acronyms

CMHP	Common Message Handling Protocol
CRC	Cyclic Redundancy Check
CTE	Conformance Test Executive
FAA	Federal Aviation Administration
FNTB	FTI National Test Bed
FTI	FAA Telecommunications Infrastructure
GPS	Global Positioning System
ICD	Interface Control Document
IP	Internet Protocol
IRD	Interface Requirement Document
LAN	Local Area Network
NAS	National Airspace System
NAT	Network Address Translation
RFS	Request For Service
SUT	System Under Test
TCP	Transmission Control Protocol
Tech Center	William J. Hughes Technical Center
TPP	Telecommunications Program Plan

4 GENERAL GUIDANCE

4.1 Purpose

CMHP is an application-level protocol that conceptually "sits" between the application and the TCP/IP stack. Two CMHP-based applications initially communicate in a client-server relationship (as Figure 4-1 depicts) with one CMHP application configured to be a CMHP client application, and the other as a CMHP server application.

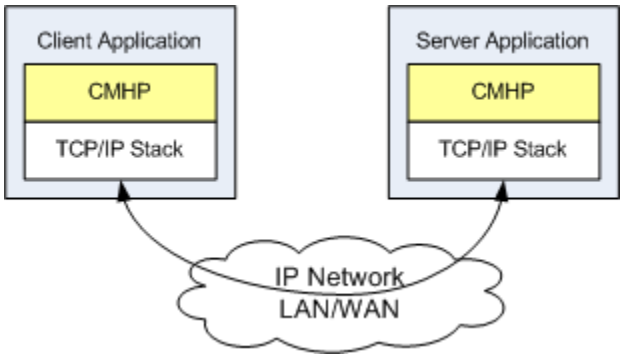


Figure 4-1 CMHP Overview

Before any application-level data can be exchanged, the CMHP client application has to successfully register for service with the CMHP server application. Once registered, the two applications operate in a peer-to-peer relationship where either side has the ability to send and receive application-level data messages, and each side provides positive acknowledgment back to the other that it has successfully received and processed the other side's data messages. In addition, either application has the ability to gracefully notify the other to terminate service before closing the TCP socket.

CMHP enables the exchange of application data with assurance that the data sent has been successfully received and processed by the far-end application. It has been developed to provide several management functions to support TCP socket-based applications.

The key management functions provided by CMHP are:

a. User Registration – In IP-based networks, the use of proxy servers and Network Address Translation (NAT) devices have the ability to modify the source IP address field, thereby making it impossible for end systems to uniquely identify the originating system of the TCP socket. CMHP provides an application-level message exchange mechanism that enables a CMHP server to uniquely identify the CMHP client, thereby enabling the CMHP server to determine whether to accept or deny service.

b. Message-Delivery Assurance – IP-based protocols are not message-based, but are stream-oriented. There are no robust mechanisms in place in the underlying IP protocols to provide message-delivery assurance at the application level. CMHP provides a built-in sliding window protocol that enables a system to acknowledge the receipt of application-level data messages back to the sender.

c. Keep-Alive – Although there is an acknowledgment within TCP, there is a requirement at the application-level to ensure that TCP sockets are still active. This cannot be accomplished without transmitting something across the socket. CMHP provides an application-level Acknowledgment message that applications can use after a period of communications inactivity to ensure that the TCP socket is still active by forcing the far-end application to reply.

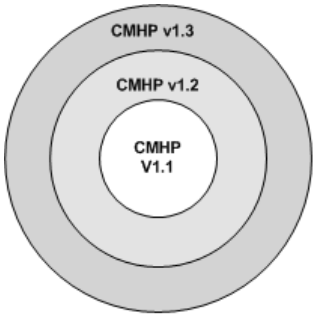
CMHP requires that a common header be used for all application-level message exchanges, regardless of whether they are management or data messages. The CMHP header has two key fields: the Message Length and the Message Type fields.

a. A Message Length field is needed by any application using TCP/IP sockets, and is the first field of all messages sent across the interface. As TCP/IP is a streams-based protocol, there is no concept of messages. Applications generally use a length field to determine the number of bytes to associate with an incoming message and read that number of bytes off the TCP/IP stack before passing the complete message up for further processing. The next byte(s) read off the TCP/IP stack after that should be the length field of the next message.

b. CMHP specifies a field in the header to be used to define the type of message being processed. This can be either one of several ***CMHP management messages*** or a ***CMHP data message*** that contains application-specific data. Each type of CMHP message should have a unique message type value.

4.2 General History

This section provides a detailed description of the protocol and the application-level CMHP messages that are supported. There are three variants (versions) of the CMHP protocol (1.1, 1.2, and 1.3), with each one encompassing the prior version.



a. CMHP v1.1 is the core version of the protocol that provides most of the CMHP functionality.

b. CMHP v1.2 encompasses all of CMHP v1.1 functionality, supports a new feature (application-level flow control) as well as adding some flexibility for message exchange during the registration process.

c. CMHP v1.3 encompasses all of CMHP v1.2 (and therefore all of CMHP v1.1), adds no additional features, but includes additional validation checks on the CMHP header.

Table 4-1 lists the feature sets and the versions under which they are supported.

Table 4-1 Feature Set Support

Feature	CMHP v1.1	CMHP v1.2	CMHP v1.3
Registration	X	X	X
Stop Service	X	X	X
Message Delivery Assurance	X	X	X
Keep-Alive Mechanism	X	X	X
Application-Level Flow Control		X	X

4.3 Functional Description

4.3.1 Registration

Registration is the process in which a client sends a request-for-service to a server before the exchange of information can begin. Under CMHP, the roles of client and server are configurable and agreed upon between two systems during an up-front agreement (such as the FAA's RFS process). This CMHP role is independent of which side initiates the TCP socket connection. A system can be assigned a server role with one user, but can be a client with another user; that is, the CMHP role should be configurable on a per-socket basis.

Regardless of which system establishes the TCP socket connection, the first CMHP message sent over this interface is the Registration Request, which is a CMHP management message. This message is only sent by the CMHP client and contains either one or two identification fields (that are agreed upon during the RFS process). The CMHP server application uses this information to identify the incoming system. In response to the incoming Registration Request message, the CMHP server system responds with a Registration Response (another CMHP management message) indicating if the Registration Request has been accepted.

If a CMHP server system rejects a Registration Request, it sends a Registration Response message back to the CMHP client (indicating the reason why the registration was refused), and then closes the TCP socket.

When a TCP socket connection is established, the CMHP server should wait a configurable amount of time (*Registration Timer)* for a Registration Request message. If one is not received within this timeframe, the CMHP server sends a Stop Service Notification (another CMHP management message) and closes the socket. If the first message received by the CMHP server across this interface is not a Registration Request message, the CMHP server sends back a Stop Service Notification message and closes the socket.

A CMHP client application should wait a configurable amount of time (*Registration Timer*) for a Registration Response message. If one is not received, it sends a Stop Service Notification message and initiates socket-closing procedures.

If the CMHP server application detects any error with any of the CMHP header fields of the Registration Request message, the CMHP server sends back a Stop Service Notification message with the appropriate error code and closes the socket.

4.3.1.1 Registration under CMHP v1.2 and Beyond

In CMHP v1.1, the first message to be sent over the TCP socket is a Registration Request message, which the CMHP client sends. Starting with CMHP v1.2, this has been loosened to enable the CMHP client to send a Stop Notification Request as the first message across the interface. However, this message should only be sent when the CMHP server is responsible for establishing the TCP socket connection to the CMHP client, and the client application is not ready to initiate the registration process. The Stop Notification Request sent by the CMHP client contains the status code (see Table 5-7 for a list of status codes) indicating this condition and when the CMHP server application receives this Stop Notification Request, it closes down the socket and should not attempt to re-establish the socket connection to the CMHP client until an agreed-upon time has passed.

In a similar scenario, when a CMHP client establishes the socket connection to a CMHP server, and issues the normal Registration Request message, the CMHP server may not be ready to support the incoming request and can now respond with a Stop Notification Request message, which contains an abnormal stop indicating the reason why the Registration Request has been refused. An example of this case would be when a CMHP client attempts to establish a connection to a backup server, instead of the

primary server. The backup server can now respond back, indicating that it is in a backup state and the client should re-attempt to connect to the primary server.

4.3.2 Stop Service

The stop service mechanism provides a graceful application-level notification for either side to use as a precursor to closing the TCP socket. A CMHP application uses this mechanism to close the connection during normal and abnormal conditions.

It is strongly recommended that CMHP applications always perform a graceful close of the TCP socket. A graceful close is an orderly shutdown process of the socket that requires all data transmitted in both directions to be acknowledged (by TCP) before the connection may be closed. A graceful close of the TCP socket allows for the far-end system to process any Stop Service Notification message before the socket is closed.

4.3.2.1 Normal Shutdown Processing

The transmission of a Stop Notification Request message (with a reason code less than 0x1000) by a CMHP application indicates that it is not transmitting any more application messages, and is requesting a graceful shutdown for normal operational reasons. The application should then wait for a configurable period (*Shutdown Timer*) to receive a Stop Notification Response message back. On receipt, or if the period for waiting expires, the CMHP application should initiate socket-closing procedures.

The CMHP application that initiates the stop service can implement one of the two following options on how to handle subsequent incoming messages from the far-end system.

a. It can either ignore all subsequent incoming messages (whether they are valid or not) and wait for the Stop Notification Response (or associated timer to expire), or

b. It can process the incoming messages, acknowledge them as normal, and then wait for the Stop Notification Response message (or associated timer to expire).

The CMHP application receiving a Stop Notification Request message (indicating a normal shutdown) should issue a Stop Notification Response (in response), but the transmission of this message also indicates that it will not send any subsequent CMHP messages and will initiate its own socket-closing procedures. However, the transmission of the Stop Notification Response message can be deferred until any and all messages in the process of being transmitted are sent and acknowledged by the far-end system.

4.3.2.2 Abnormal Shutdown Processing

If either side has a major problem with processing any CMHP message, it should send a Stop Service Notification message indicating the problem before initiating socket-closing procedures. The Stop Service Notification message provides the reason for the socket being shutdown, and contains a Reason Code (found in the Status field) whose value is greater than or equal to 0x1000. An optional free-text area can also be used to provide additional system-specific information. The recipient of a Stop Service Notification message, indicating an error condition, should not respond with a Stop Service Notification Response message but should initiate socket-closing procedures.

Table 5-7 defines a list of specific status codes that must be used where applicable. If this table does not provide a status code that is pertinent, then the status codes 0x1007, 0x100E, or 0x1013 can be used, but text must be put into the optional field to supplement the use of any of these three codes. Note: if in doubt, review the conformance test plan (Section 5.6) for further insight on applicable use of status codes.

4.3.3 Message-Delivery Assurance Mechanism

The CMHP message-delivery assurance mechanism provides positive feedback that the far-end system has acknowledged the receipt of (and assumed responsibility for) one or more CMHP data messages. The premise is that a CMHP application that is receiving messages should process them and acknowledge them promptly. The feedback mechanism uses two fields in the CMHP header: the Message Sent Count M(s) and the Message Received Count M(r).

4.3.3.1 Message Sent Count

Each system maintains a Modulo-256 count for all CMHP data messages sent across the socket connection. The count is transmitted as part of the header (the M(s) field) for all messages sent across the socket connection. The count is initialized to zero upon socket establishment, and each CMHP data message is sequentially numbered to a maximum value of 255. The count then cycles back to zero. The first CMHP data message sent across the interface has an M(s) value of zero, and CMHP applications should increment the count after processing an outgoing CMHP data message. This means that all CMHP ***management*** messages always indicate the number of the next CMHP data message to be sent, and all CMHP ***data*** messages reflect their own number.

Each system maintains its own configurable transmit window so that while this transmit window is open, it enables the system to transmit CMHP data messages. The size of the transmit window can be between 1 and 255, and is a count of the number of CMHP data messages that have been sent, but have not been acknowledged by the far-end system. If a system fills its transmit window, it cannot not send any additional CMHP data messages until the far-end system sends messages back that acknowledge the receipt of one or more of those sent messages. The far-end system acknowledges messages by sending CMHP data or management messages that contain an M(r) field with the value of the message it last processed (see Section 4.3.3.2 for more details).

If no acknowledgment is received after a system-configurable period of time (*Poll Response Timer*) (after the last message in the transmit window was sent), the local application should prompt the far-end application for a reply by sending an Acknowledgment message with the Poll flag set. If there is still no response received within a specified timeframe (*Poll Response Timer*), then the system has the flexibility to repeat this polling mechanism by re-issuing an Acknowledgment message with the Poll flag set. This Acknowledgment message is re-sent periodically one or more times until a message is received that contains an updated M(r) value. If the limit for retransmitting these Acknowledgment messages is reached, then the system issues a Stop Service Notification message (indicating this error condition) and initiates TCP socket-closing procedures.

Note that instead of waiting to send the first Acknowledgment message with the Poll flag set, a system can set the Poll flag in the last CMHP data message sent before the transmit window is closed. As an example, if a system has a transmit window of three, it can either send three data messages, wait, and then send the first Acknowledgment message with the Poll flag set; or it can set the Poll flag on the third data message.

4.3.3.2 Message Received Count

Each system maintains a Modulo-256 count for all CMHP data messages it receives. The system uses this count to acknowledge to the sending system (the far-end system) that it has successfully received and processed a specific message or group of consecutive messages. This count is transmitted as part of the CMHP header (the M(r) field) in all CMHP data and management messages sent back to the far-end system.

The value reflected in the M(r) field value is the number of the next CMHP data message that the system is expecting to process. If the M(r) value seen on incoming messages is equal to the M(s) value (for the

next message to be sent), it indicates there are no outstanding/unacknowledged messages. (Note: M(r) should have a value of zero at socket initialization.)

A system should change the M(r) value when an incoming CMHP data message has been successfully processed. The system can process one or a group of consecutively numbered messages, modifying the M(r) field to the value of the next unacknowledged message, and then sending the updated M(R) back to the far-end system.

When a system processes an incoming M(r) value of x, it assumes that all messages numbered up to and including x-1 have been successfully received and processed by the far-end system.

A system acknowledges receipt of incoming messages in two ways:

a. The first is to update the M(r) field in any CMHP management or data message it sends back across the interface.

b. The second is if there are no data messages waiting to be sent, a system sends a CMHP Acknowledgment message updating the M(r) field.

Figure 4-2 depicts a simple interaction where a CMHP client system (that has a transmit window of three) sends five messages to a CMHP server system, and then closes the socket. (Note the TCP socket establishment and closure are not shown.)

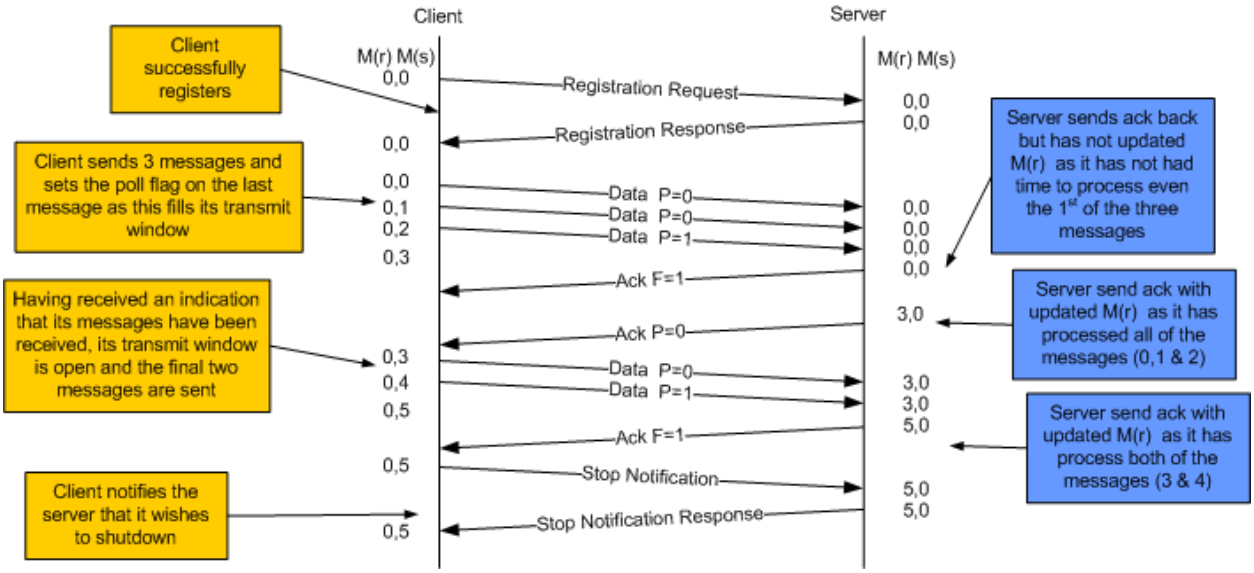


Figure 4-2 Example of M(s) and M(r)

In the above example, the client system has a transmit window of three (that is., the maximum number of unacknowledged messages it can transmit before it requires an acknowledgment). After the client system successfully registers, it sends the first three messages and forces a far-end acknowledgment by setting the Poll flag in the header of the third message. In this example, although the server has received the first three messages, it has not had time to process any of them and replies as such with an Acknowledgment message with the Final flag set as well as setting the value of the M(r) field to zero. However, after a few seconds, when the server has processed all three incoming messages, it sends an acknowledgment with the M(r) value set to three (indicating the number of the next unacknowledged message it expects to process). On receipt of the acknowledgment, the client's transmit window re-opens and the client system

is then able to transmit its final two messages. The server is able to process these messages and send back an Acknowledgment message with the Final flag set and with an M(r) value set to five to indicate that the next message it is looking to process is five.

4.3.4 Keep-Alive Mechanism

The CMHP keep-alive mechanism is used to periodically probe the other end of a connection when the connection has been idle for a configurable amount of time (*Keep-Alive Timer*). This mechanism is used by the system that is responsible for establishing and maintaining the TCP sockets (typically, the client); however, in reality, the system that has the smaller inactivity timer always triggers the keep-alive mechanism.

When a system's Keep-Alive Timer expires (that is, the condition has been detected that no messages have been sent or received on a connection), a system sends an Acknowledgment message with the Poll flag set. If the originator (of the Acknowledgment message) does not receive a response, it resends the Acknowledgment message and repeats this cycle for a configurable number of attempts. If the limit of attempts is reached, the system transmits a Stop Service Notification message and initiates socket-closing procedures.

When a system receives an Acknowledgment message with the Poll flag set, it replies with either a data message that it has queued up ready for transmission, or with an Acknowledgment message. In either case, the Final flag is set.

4.3.5 Application-Level Flow Control

CMHP v1.2 introduced a new feature, application-level flow control, which enables one application to notify the other of its ability to receive data. The Flow Control flag has been defined (in the CMHP header) that when clear (set to zero by the local application), it indicates that the local application is able to receive data. When set (to 1), it indicates that the local application is temporarily unable to process any more incoming data.

Any time after a successful registration when CMHP is notified by its local application of a change in the application's ability to receive incoming CMHP data messages, the CMHP application should change the Flow Control flag accordingly. The change to the flag should trigger CMHP to notify the far-end application in one of two ways.

a. If there is any type of message in the queue ready to be sent to the far-end, then the CMHP application should send that message (in the usual manner), but reflects the new state of the Flow Control flag;

b. Otherwise, if there is nothing queued, the CMHP application sends an immediate CMHP Acknowledgment message, which reflects the new state of the Flow Control flag.

The Flow Control flag can be set on any type of CMHP message (including the Registration Request and Registration Response messages) and should remain in effect (that is, will be set) on all subsequent CMHP messages. The same is true for the flag being cleared; it can be cleared on any type of CMHP message and should remain in effect (cleared) on all subsequent CMHP messages.

When the local application receives a CMHP message with the Flow Control flag set, it should cease transmission of all subsequent CMHP data messages until the Flow Control flag is cleared by the far-end application. Note, however, this flag has no impact on the local application's ability to transmit CMHP management messages, nor on the local application's ability to continue to receive CMHP data messages from the far-end application.

When a local application sets the Flow Control flag, it may still receive a few incoming data messages. This condition typically occurs when these messages are in transit from the far-end application before the far-end application is able to receive and process the Flow Control flag. How the local application handles these last incoming messages is beyond the scope of this handbook.

4.3.6 Additional CMHP Error Handling

4.3.6.1 CRC Validation

CMHP supports a 32-bit Cyclic Redundancy Check (CRC) field in the CMHP header. Before transmission of all CMHP messages, the CRC field is zeroed out and a standard CRC-32 calculation (CRC-32-IEEE 802.3) is performed across the entire CMHP message. The resulting checksum is stored in the 32-bit CRC field of the CMHP header.

On receipt of all incoming messages, the value in the CRC field is saved, the field zeroed out, and the CRC calculated and compared against the saved value. If the comparison fails, the system should issue the Stop Service Notification message with the applicable error code and initiate socket-closing procedures.

4.3.6.2 CMHP Header Validation

The CMHP applications must perform validation on specific fields in the CMHP header every time an incoming message is received. For any problem detected, the only way to recover is to send a Stop Service Notification message that indicates the error and to initiate socket-closing procedures. Table 5-7 defines a number of predefined error codes covering the anticipated error conditions.

At a minimum, the following CMHP header validations are to be performed:

a. Verify that the CMHP version is supported.

b. Verify that the Message Type field contains a valid value for the application.

c. Verify that the Message Type field contains a value that is valid for the current protocol state.

d. Verify that the Message Length is valid for the Message Type.

e. Verify that the Major and Minor fields are valid.

f. Verify the M(s) and M(r) values.

g. Verify that the status field has a value valid for the Message Type.

4.3.6.3 Additional CMHP Validation under CMHP v1.3

The following validation checks on the CMHP header are to be performed under CMHP v1.3:

a. Verify that Major and Minor versions do not change after registration. Note that the CMHP client sets the CMHP version on the outgoing Registration Request message. A CMHP server application has the option to either statically define the CMHP version for a specific CMHP client and verify every incoming CMHP message against that statically defined value, or to dynamically set the value when receiving the incoming Registration Request message. Note that the system-specific IRD/ICDs should define whether they support the static or dynamic nature of these fields.

b. Verify that the remote application's Source Location Identification is valid. This value can either be statically defined, or can be dynamically set by an incoming Registration Request (for a CMHP server application) or by an incoming Registration Response (for a CMHP client

application). Note that the system-specific IRD/ICDs should define whether they support the static or dynamic nature of these fields.

c. Verify that the Flag field only uses the bit fields associated with the selected CMHP version.

d. Verify that the Spare fields are set to binary zero.

4.3.7 Byte-Order Mandate

Byte-order handling is the ability to convert multi-byte numeric fields from host byte-order to network byte-order for outbound messages and to convert from network byte-order to host byte-order on inbound messages. This functionality enables CMHP applications to send and receive messages over IP-based communication networks regardless of the “endianness” of either system.

The CMHP application is responsible for the byte-order handling of the CMHP header for all messages and the CMHP payload for CMHP management messages only. Applications are responsible for the byte-order handling of the CMHP payload for all CMHP data messages.

4.3.8 TCP/IP Partial Read Timeout

Although CMHP can reside on top of any underlying communications protocol, the anticipation is that TCP/IP will be the prevalent protocol. It is strongly recommended for time-sensitive applications that when using TCP/IP, the handling of the TCP/IP partial read timeout be leveraged to detect missing incoming data rather than using CMHP’s Keep-Alive mechanism.

5 DETAILED GUIDANCE

This section defines the CMHP functionality supported, message formats, and message structures.

5.1 Protocol Stack

CMHP is an application-layer protocol that is used to exchange application data using TCP/IP, as per the Internet Protocol stack that Figure 5-1 shows.

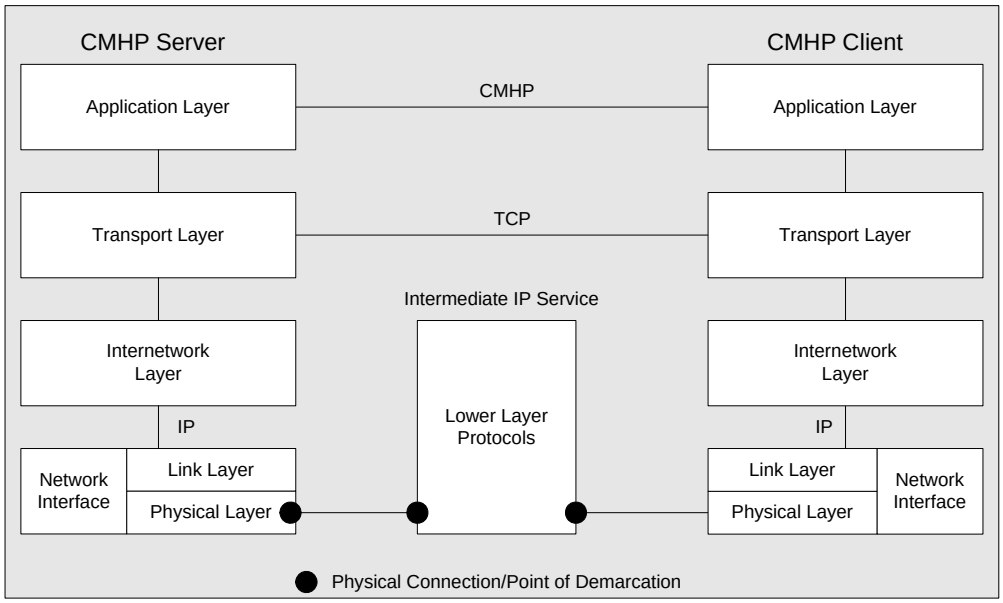


Figure 5-1 Protocol Stack

5.1.1 Transport Layer - TCP Segment Format

A TCP segment consists of the TCP header and TCP data fields (as Figure 5-2 depicts) that reside within the data field of an IP datagram. The definition of the individual fields within the TCP header is contained in RFC 793 (IETF STD-7). The TCP header field is 20 bytes in length.

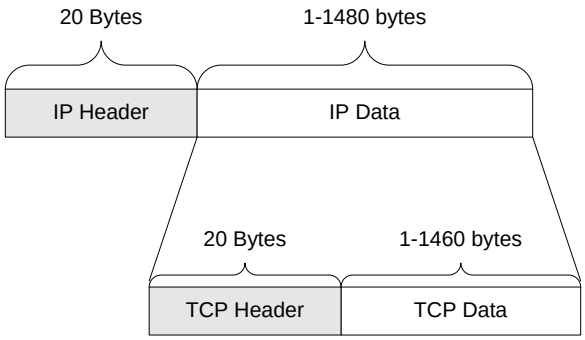


Figure 5-2 Standard TCP Segment Structure

5.1.2 Application Layer – CMHP Message Format

A CMHP message consists of the CMHP header and an optional CMHP payload (as Figure 5-3 illustrates). A CMHP message can reside anywhere in the TCP data field of a TCP segment. There could be multiple CMHP messages in one TCP data field or a CMHP message could span multiple TCP data fields, which would mean that not all TCP data fields include a CMHP header.

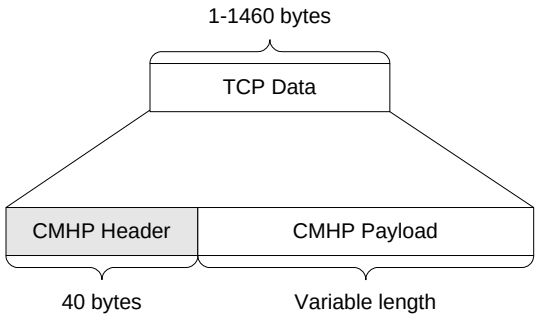


Figure 5-3 CMHP Message Format

CMHP supports CMHP management messages, which the rest of this section describes in detail, and CMHP data messages. System-specific Interface Requirement Documents and Interface Control Documents (IRDs and ICDs) define CMHP data messages.

5.2 CMHP Header

All CMHP messages (data and management) transmitted over a TCP/IP socket connection should include a CMHP header as Table 5-1 defines.

Table 5-1 CMHP Header in v1.1, v1.2, and v1.3

Name	Length (Bytes)	Format	Description
Message Length	4	Numeric	Length of the CMHP message including the CMHP header
Message Type	2	Numeric	Defines the type of message – for example, AFTN, WMO, OMO, Keep-Alive, Registration Request
Major Version	1	Numeric	Set to 0x01
Minor Version	1	Numeric	Set to 0x01, 0x02, or 0x03
M(s)	1	Numeric	Message Sent Count
M(r)	1	Numeric	Message Received Count
Flags	1	Bit	Bit 0 – Poll flag Bit 1 – Final flag Bit 5 – Flow Control flag (v1.2 only)
Spare	1	Numeric	Set to 0x0
Status	2	Numeric	This field provides supporting information based on the Message Type – Default value 0x0000
Timestamp - Minutes	2	Numeric	Minute of the Day message sent (0 – ((24*60)-1)
Timestamp - Seconds	4	Numeric	Microsecond of the Minute (0 – (60*100000)-1)
Source Location ID	8	ASCII	Identifier used to depict the sender of the message. Pad with binary zeros
Spare	8	ASCII	To be used for future options – Set to 0x0.
Checksum	4	Numeric	32-bit CRC

5.2.1 Host-Network Byte-Ordering Requirements

As per Section 4.3.7, the CMHP application is responsible for converting the following multi-byte numeric fields in the CMHP header:

a. Message Length

b. Message Type

c. Status

d. Timestamp fields

e. Checksum.

5.2.2 Message Length

The Message Length field is the first four bytes of the CMHP header and contains the length of the CMHP message. The length field includes the total number of bytes of the CMHP header (including the Message Length field) and the (optional) CMHP payload.

5.2.3 Message Types

CMHP supports the following CMHP management message types. See Section 5.4 for the list of currently defined CMHP data message types.

Table 5-2 CMHP Management Message Types

CMHP Management Messages	Message Type Value	Message Length (bytes)
CMHP Management Requests/Notifications		
Registration Request	0x0002	72 or 88
Stop Service Notification	0x0003	40 – 296
CMHP Management Request/Response		
Acknowledgment	0x0040	40
CMHP Management Responses		
Registration Response	0x0082	40
Stop Service Notification Response	0x0083	40

5.2.4 Version Fields

All headers contain a version number representing the Major.Minor format. The major number should only change if there is an update to the structure of the header, which results in a change to the header size. Therefore, the major version should always be associated with the same size header. The v1.x series header is 40 bytes. The minor field should change if new fields are defined in the spare areas of the header, or if new functionality is added.

5.2.5 Status Field

The use of the Status field is specific to each type of CMHP message. For those CMHP management messages where the Status field has no relevance, it is set to 0x0000 on output and is ignored on input for CMHP v1.1 or v1.2. However for v1.3, this field must be verified to be non-zero and is detected and reported by sending a Stop Notification Message with the appropriate status code (see the status codes for CMHP v1.3 in Table 5-6).

5.2.6 Timestamp Fields

The current concept is to provide a mechanism to determine network latency in both directions, but for it to be useful, systems using this feature need to be synchronized, that is, based on GPS or quantum clocks.

5.2.7 Source Location Identification Field

The Source Location Identification field is used to depict the source of the message.

5.2.8 Message Sent Count Field

Every message transmitted across the interface is numbered in each direction. A Modulo-256 scheme is used for the sequence-numbering scheme. The message sequence cycles through the whole range from 0 to 255. The first message sent after the establishment of the socket connection has the send-sequence number set to zero.

5.2.9 Message Received Count Field

This byte field represents the last acknowledged message received by a system. A Modulo-256 scheme is used for the sequence number. The message sequence cycles through the whole range from 0 to 255. The initial value upon socket establishment is zero.

5.2.10 Flags

5.2.10.1 CMHP v1.1

In CMHP v1.1, this byte field defines the use of only two flags: the Poll and Final flags. These flags support the message-delivery assurance mechanism. The Poll flag is used as a command indicating that the far-end has to respond back. The Final flag is used to acknowledge the Poll flag. (Note: Bit 0 denotes the least-significant bit.) A flag is set when it has a value of 1, and it is clear when it has a value of 0. The other six unused flags are set to 0.

The Poll and Final flags are clear for Registration Request or the Registration Response messages. However, these fields should be ignored on input, that is, no validation needs to be performed on these fields.

The Poll and Final flags can be set for the Stop Notification Request (when the status code < 0x1000), as this message indicates a normal shutdown and requires receipt of a Stop Notification Response message before the socket is closed.

The Final flag can be set for the Stop Notification requests (when the status code >= 0x1000) indicating an error condition. Setting the Poll flag in this case makes no sense, and should be ignored on input.

For Stop Notification Response messages, although setting the Final flag is valid, setting the Poll flag is not. The Poll flag should be ignored on input.

5.2.10.2 CMHP v1.2

In CMHP v1.2, this byte field defines the use of three flags. The Poll and Final flags are used as per CMHP v1.1. The third flag, defined in CMHP v1.2, is the Application-Level Flow Control flag. This is a pass-through flag that is set and cleared by the application to denote its ability to receive data (flag is set to 0) and when the application is unable to receive data (flag is set to 1). The other five flags are to be set to 0 on output and should be ignored on input.

5.2.10.3 CMHP v1.3

The Poll and Final Flags must be cleared on Registration Request and Registration Response messages. However, if the Poll or Final flag is set on a Registration Request or a Registration Response, the condition must be detected and reported by sending a Stop Notification Message with the appropriate status code (see the status codes for CMHP v1.3 in Table 5-6).

5.2.11 Spare Fields

The Spare fields within the header are used to pad structures to 4-byte boundaries, to provide flexibility for future capability, and to ensure the CMHP header is 40 bytes in length. These spare fields must be set to binary zero. Under CMHP v1.3, a non-zero condition must be detected and reported by sending a Stop Notification Message with the appropriate status code (see the status codes for CMHP v1.3 in Table 5-6).

5.2.12 Checksum Field

This field contains a standard CRC-32 (CRC-32-IEEE 802.3) value, which is calculated on the CMHP header and CMHP payload. Note that the checksum field is set to binary zero before the calculation is performed, and is replaced with the calculated CRC value on transmission.

5.3 CMHP Management Messages

CMHP supports the following types of CMHP management messages:

a. Acknowledgments

b. Registration Requests and Responses

c. Stop Notifications and Responses.

5.3.1 Acknowledgment Message

The Acknowledgment message consists solely of the CMHP header with the Message Type field set to 0x0040. It is used for the Keep-Alive mechanism, as well as being one of the ways to acknowledge the receipt of one or more incoming CMHP data messages, and one of the ways to indicate a change in application-level flow control status.

If the acknowledgment is being forced (by the receipt of an incoming message with the Poll flag set), the local system must respond with the Final flag set; otherwise, for all other conditions, the Final flag is set to zero.

If a system receives an Acknowledgment message with the Final flag set, but the system has no outstanding Poll flag set, the Acknowledgment message is processed normally and the Final flag is ignored.

If a system receives an Acknowledgment message with the Poll flag clear, but the system has an outstanding Poll flag set, the system should ignore the Acknowledgment message.

For CMHP v1.2 and later, whenever the local application changes the value of the Application-Level Flow Control flag, an Acknowledgment message is sent to notify the far-end of the new status of the flag, unless there is a pending message to be sent, in which case, that pending message can be used to reflect the new status of the Flow Control flag.

5.3.2 Registration Request Message

The Registration Request message consists of the CMHP header (with the Message Type field set to 0x0002) and a CMHP payload that consists of either one or two supporting fields.

The Primary Identifier field is mandatory and is a fixed length 32-byte field, and if the primary identifier itself is less than 32 ASCII characters, then the remainder of this field is filled with binary zeros (0x0).

The Secondary Identifier field is optional, and is not sent as part of the Registration Request message if the CMHP server application does not require a secondary identifier. If required, this field is a fixed length, 16-byte field, and if the secondary identifier itself is less than 16 ASCII characters, then the remainder of this field is filled with binary zeros.

Table 5-3 Registration Request Message

Name	Length	Format	Description
Primary Identifier	32	ASCII	Mandatory identifier
Secondary Identifier	16	ASCII	Optional identifier

5.3.3 Registration Response Message

The Registration Response message is sent by the CMHP server application in response to a Registration Request message. It should consist only of the CMHP header, with the Message Type field set to 0x0082 and the Status field, which should contain one of the following values.

Table 5-4 Registration Response Message Status – Response Codes

Status Field Value	Meaning
0x0001	Registration Successful
0x1001	Registration Failed – Unknown Primary Id
0x1002	Registration Failed – Invalid Secondary Id
0x1003	Registration Failed – Access Barred

5.3.4 Stop Service Notification Message

The Stop Service Notification message consists of the CMHP header (with the Message Type field set to 0x0003) and an optional CMHP payload that consists of a free-text variable-length field to be used to provide additional information/reason for stopping the service.

Table 5-5 Stop Service Notification – Optional Field

Name	Length	Format	Description
Text	256	ASCII	Free text area to provide additional error condition information

If the Status field (in the CMHP Header) contains a value that is the range 0x0001-0x0FFF it indicates a "normal" shutdown condition and requires the far-end system to respond back with a Stop Notification Response message. Table 5-6 defines the one predefined status code; the range allocated for future CMHP use, and the range allocated for system-specific conditions.

Table 5-6 Stop Service Notification Message – Status Codes for Normal Conditions

Status Value	Meaning
0x0001	Normal operational request for shutdown
0x0002 – 0x01FF	Reserved for future CMHP use for non-error conditions
0x0200 – 0x0FFF	System-defined shutdown codes for non-error conditions.

If a system-specific status code is used, then the optional text field (see Table 5-5) must contain supporting text.

If the Status field (in the CMHP Header) contains a value that is in the range 0x1000-0xFFFF, it indicates an abnormal condition has occurred and no Stop Notification Response message is required to be returned. Table 5-7 lists pre-defined status codes that CMHP applications must use when applicable. However, there is a range of status codes available to support system-specific conditions (0x2000-0xFFFF). If a system-specific status code is used, then the optional text field (see Table 5-5) must contain supporting text.

Table 5-7 Stop Service Notification Message – Status Codes for Abnormal Conditions

Status Value	Meaning
0x1006	Poll response timeout
0x1007	Illogical condition – catch-all
0x1008	Received message length not valid
0x1009	Received message type not defined
0x100A	Received message has unexpected version
0x100B	No local buffers
0x100C	Received message byte count error
0x100D	Non-registration message received when not registered
0x100E	System in wrong state
0x100F	Bad checksum on received message
0x1010	Registration request timeout
0x1011	Registration response timeout
0x1012	Message larger than max allowed for this interface
0x1013	Interface status not ok for received message
0x1014	Received M(s) is not expected
0x1015	Received M(r) out of valid range
0x1019	Partial Read timeout
0x101A	Received message invalid as declared by application
0x2000 – 0xFFFF	System-specific errors
Added under CMHP v1.3	
0x101B	Received message has illegal flags set for CMHP version
0x101C	Invalid Source Location field detected
0x101D	First spare field contains non-zeros
0x101E	Second spare field contains non-zeros
0x101F	Invalid status code detected
0x1020	Registration message received with Poll flag set – illegal condition
0x1021	Registration message received with Final flag set – illegal condition

5.3.5 Stop Service Notification Response Message

The Stop Service Notification Response message consists of the CMHP header with the Message Type field set to the value 0x0083. This message is only sent back in response to a Stop Service Notification message that has a Status field value of less than 0x1000.

5.4 CMHP Data Messages

All CMHP data messages are specific to the CMHP-compliant applications and should, therefore, be defined in the associated system-specific IRD and ICDs.

5.5 CMHP Timers

This section describes a set of timers associated with the implementation of this protocol.

5.5.1 Keep-Alive Timer

The Keep-Alive Timer tracks the time an application tolerates inactivity in both directions over a connection before sending an Acknowledgment message with the Poll flag set, forcing the far-end to respond with an Acknowledgment message with the Final flag set. The recommendation is for the side that initiates the socket connection be the one with the smaller Keep-Alive Timer to detect and recover from a loss of the underlying TCP connection.

5.5.2 Partial Read Timer

This timer is a function of the underlying TCP protocol, and is the time the application waits after receiving at least the first byte of a message until receiving the complete message. If this timer expires, the system should send a Stop Notification Request message with a code of 0x1019. The system should then initiate socket-terminating procedures.

5.5.3 Poll Response Timer

The Poll Response Timer is the time an application waits after sending a message with the Poll flag set to receiving a response that includes an updated M(r) value.

5.5.4 Registration Timer

For CMHP servers, this timer is associated with the time between a TCP socket being established and the time to wait for a Registration Request message to be received from the CMHP client. If the Registration Timer expires, the CMHP server should send a Stop Notification Request message with a code of 0x1010. The CMHP server then initiates socket-terminating procedures.

For a CMHP client, this is the time between sending a Registration Request message and receiving a Registration Response message. If the Registration Timer expires, the CMHP client sends a Stop Notification Request message with a code of 0x1011. The CMHP client then initiates socket-terminating procedures.

5.5.5 Shutdown Timer

This is the time after sending a Stop Notification Request with a status code <0x1000 to receive a Stop Notification Response. If this timer expires, then the application should initiate socket-closing procedures.

All CMHP applications should wait a minimum time after sending a Stop Notification message with an error code (>=0x1000) before initiating socket-closing procedures to allow the Stop Notification message to be sent by the local stack across the interface to the far-end system.

5.6 Conformance Test Plan for CMHP

This section describes the CMHP Conformance Test Plan. The CMHP Conformance Test Plan provides a starting point for the detailed planning required to test any CMHP implementation to determine its conformance to this handbook. This section provides an overview to the test cases, an understanding of the overall goals, and the strategy to accomplish them.

5.6.1 Scope

The CMHP Conformance Test Plan documents the test scripts that have been developed as part of the CMHP Conformance Test Executive (CTE) test tool that can be run against CMHP v1.1, CMHP v1.2, and CMHP v1.3, and against an application that can be configured either as a CMHP client or a CMHP server.

5.6.2 Purpose

The purpose of the CMHP conformance tests is to ensure that any application that has developed CMHP is fully compliant with this handbook. This is accomplished by running a series of conformance test scripts against the CMHP implementation to ensure that it operates correctly during normal and abnormal conditions.

Figure 5-4 is a high-level overview of the test setup. The CMHP test tool runs test scripts that generate a stimulus to the CMHP application running on the System Under Test (SUT). The SUT responds to that stimulus and its response is verified against the list of responses that are valid for that particular stimulus.

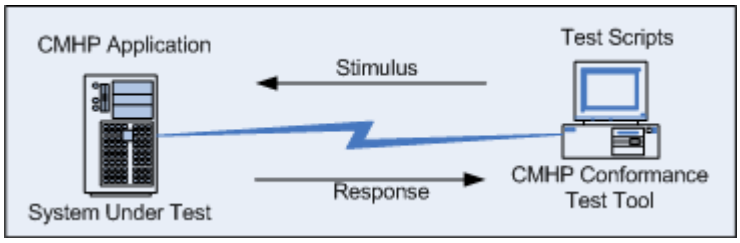


Figure 5-4 Test Setup

For example, if the stimulus is a CMHP Registration Request message, the response (under normal conditions) the SUT sends back a CMHP Registration Response. However, in a subsequent test the test tool sends a CMHP Registration Request, receives back a Registration Response from the SUT, and then sends a second Registration Request to verify that the SUT can handle this error condition.

The conformance test scripts verify three main areas of the SUT's implementation of CMHP:

- Normal CMHP protocol exchanges.
- Illegal protocol exchanges – that is, sending a valid CMHP message that is not appropriate for the current state of the protocol – such as stated above
- Corrupted messages – that is, a CMHP message whose header has been deliberately corrupted, shortened, or lengthened.

CMHP itself requires a CMHP implementation to be configured as either a CMHP client application or a CMHP server application. Some systems may implement both capabilities and act as a CMHP server with one system, but as a CMHP client with another system, as Figure 5-5 depicts.

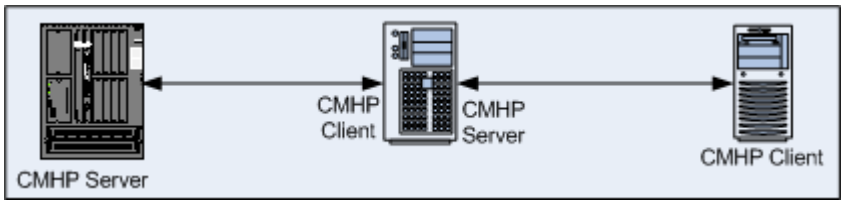


Figure 5-5 CMHP Roles

Therefore, the conformance test scripts are able to test the SUT regardless of which CMHP role it is configured as.

These CMHP conformance tests are pertinent to all versions of CMHP (v1.1, v1.2, and v1.3).

Successful completion of the CMHP conformance tests is one of the entry requirements that the SUT must complete before the SUT is allowed to test against any of the FAA test bed systems.

The CMHP conformance tests do not verify any application-level data. Application data sent and received by the test scripts as part of these tests should be ignored.

5.6.3 CMHP Test Procedures

This section describes each of the tests that constitute the baseline conformance tests for CMHP v1.1, CMHP v1.2, and CMHP v1.3.

For CMHP v1.1, the tests are broken down into the first 17 test groups as Table 5-8 reflects.

- There are 3 test groups that contain client-specific tests (24 tests).
- There are 3 test groups that contain server-specific tests (29 tests).
- The remaining 11 groups are common for both client- and server-oriented tests (93 tests).

For CMHP v1.2, the tests are broken into 19 test groups as Table 5-8 reflects.

- There are 4 test groups that contain client-specific tests (27 tests).
- There are 4 test groups that contain server-specific tests (32 tests).
- The remaining 11 groups are common for both client- and server-oriented tests (93 tests).

For CMHP v1.3, the tests are broken into 19 test groups as Table 5-8 reflects.

- There are 4 test groups that contain client-specific tests (33 tests).
- There are 4 test groups that contain server-specific tests (39 tests).
- The remaining 11 groups are common for both client- and server-oriented tests (111 tests).

Table 5-8 Conformance Test Groups

Test Overview					
Group	**SUT**	**Description**	**# v1.1 Tests**	**# v1.2 Tests**	**# v1.3 Tests**
R1	Client	Registration Request - General Tests	8	9	9
R2	Client	Registration Request - Illegal Message Tests	3	3	3
R3	Client	Registration Request - Corrupted Message Tests	13	13	19
R4	Server	Registration Request - General Tests	11	12	12
R5	Server	Registration Request - Illegal Message Tests	5	5	5
R6	Server	Registration Request - Corrupted Message Tests	13	13	20
R7	Either	Registration Violations - General Tests	2	2	2
R8	Either	Registration Violations - Corrupted Message Tests	13	13	13
A1	Either	Ack - General Tests	5	5	5
A2	Either	Ack - Corrupted Message Tests	12	12	17
S1	Either	Stop Notification Request- Corrupted Message Tests	12	12	17
S2	Either	Stop Notification Response - Illegal Message Tests	8	8	8
S3	Either	Stop Notification Response - Corrupted Message Tests	12	12	16
S4	Either	Stop Notification Response - Illegal Message Tests	1	1	1
D1	Either	Data Transfer - Corrupted Message Tests	13	13	17
D2	Either	Data Transfer - SUT Receiving Test Data	5	5	5
D3	Either	Data Transfer - SUT Sending Test Data	10	10	10
F1	Server	Flow Control - General Tests	0	2	2
F2	Client	Flow Control - General Tests	0	2	2
		Total	146	152	183

CMHP is a predominantly balanced protocol except for the Registration exchange. Only a CMHP client application can initiate the Registration process after a TCP socket has been established. After a successful Registration, either side has the ability to send and receive data and to initiate service-termination procedures.

These tests have been designed so that each test is self-contained and does not require successful prior test execution for it to run. Each test starts with a TCP socket establishment and concludes (explicitly or implicitly) with the TCP socket being closed.

5.6.3.1 R1 – Registration Request – General Tests (Client)

These tests are only valid for an SUT that is configured as a CMHP client. The purpose of this group of tests is to ensure that the SUT is able to initiate the registration process and to handle normal and error responses from the CTE test tool.

Test #	Overview	Test Purpose
R1-01	Valid Registration (PID only) and Shutdown	Verify that when the SUT sends a Registration Request containing only the PID, that it accepts the Registration Response as well as the subsequent normal Stop Notification Request.
R1-02	Registration Timeout	Verify that when the SUT sends a Registration Request containing only the PID, that the SUT then sends a Stop Notification Request with a status code 0x1011 after no response is received to its Registration Request.
R1-03	Stop Notification	Verify that when the SUT sends a Registration Request containing only the PID, but it receives a Stop Notification Request message in response, that the SUT either sends a Stop Notification Response, or a Stop Notification with a status code 0x100D.
R1-04	Bad Primary Id	Verify that when the SUT sends a Registration Request containing only an invalid PID, that the SUT closes the socket when it receives a Registration Response with a status code of 0x1001.
R1-05	Bad Secondary Id	Verify that when the SUT sends a Registration Request containing only the PID (but the CMHP server is expecting a PID and a SID), that the SUT closes the socket when it receives a Registration Response with a status code of 0x1002.
R1-06	Empty Secondary Id	Verify that when the SUT sends a Registration Request containing the PID and SID, but the SID field is "empty" (all nulls), that the SUT closes the socket when it receives a Registration Response with a status code of 0x1002.
R1-07	Valid Registration (PID & SID) and shutdown	Verify that the SUT accepts a Registration Response in response to its Registration Request containing a PID and SID, and that it handles a subsequent normal Stop Notification Request.
R1-08	Registration (PID & SID) 32 & 16 char	Verify that the SUT is able to send a 32-character PID and a 16-character SID in the Registration Request message.
New Tests added for v1.2 and Beyond		
R1-09	Stop Notification Request	Verify that when the SUT sends a Registration Request that the SUT closes the socket when it receives a Stop Notification Request with a system-specific status code and text field.

Additional Notes:

- Test R1-02 should also be used to verify the accuracy of the SUT's Registration Timer.
- Test R1-06 requires the SUT to be reconfigured to send the PID and SID fields.
- Tests R1-07 and R1-08 require the SUT to change the contents of the PID and SID fields.
- Test R1-09 is a new test for CMHP v1.2 (and beyond) systems, and verifies such cases where a system [configured as a CMHP server] might be in a backup role and accepts incoming sockets, but uses the Stop Notification Request to respond to an incoming Registration Request to indicate its state and implicitly indicate that the client should re-reroute its next socket request to a primary system.

5.6.3.2 R2 – Registration Request – Illegal Message Tests (Client)

The purpose of this group of tests is to ensure that the SUT (when configured as a CMHP client) is able to correctly handle valid CMHP messages that are not valid for the current state of the protocol.

Test #	Overview	Test Purpose
R2-01	Send Data	Verify that when the SUT receives application data in response to its Registration Request, it sends a Stop Notification Response with a status code 0x100D, 0x100E, or 0x1013.
R2-02	Send an Ack	Verify that when the SUT receives an Acknowledgment message in response to its Registration Request, it sends a Stop Notification Response with a status code 0x100D, 0x100E, or 0x1013.
R2-03	Send a Stop Notification Response	Verify that when the SUT receives a Stop Notification Response in response to its Registration Request, it sends a Stop Notification with a status code 0x100D, 0x100E, or 0x1013.

5.6.3.3 R3 – Registration Request – Corrupted Message Tests (Client)

The intent of this Test Group is to ensure that the SUT (when configured as a CMHP client) successfully handles the set of conditions where it receives a corrupted Registration Response message in response to its Registration Request message. In all cases, the SUT shall issue a Stop Notification Request message with the appropriate status code.

Test #	Overview	Test Purpose
R3-01	Short Message Length	Verify that when the SUT receives a Registration Response message with a message length less than 40 bytes, it issues a Stop Notification Request with the status code 0x1008, 0x100C or 0x1019.
R3-02	Long Message Length	Verify that when the SUT receives a Registration Response with a message length exceeding the number of bytes sent, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
R3-03	Invalid Message Type	Verify that when the SUT receives a Registration Response message, but the Message Type field contains an invalid value, it issues a Stop Notification Request with the status code 0x1009, 0x101A, or 0x100D.
R3-04	Invalid Major Id	Verify that when the SUT receives a Registration Response message with an invalid Major Id field, it issues a Stop Notification Request with the status code 0x100A.
R3-05	Invalid Minor Id	Verify that when the SUT receives a Registration Response message with an invalid Minor Id field, it issues a Stop Notification Request with the status code 0x100A.
R3-06	Invalid Status	Verify that when the SUT receives a Registration Response message with an invalid status code, it issues a Stop Notification Request with the status code of 0x1007 or 0x101F.
R3-07	Invalid Checksum	Verify that when the SUT receives a Registration Response message with an invalid CRC field, it issues a Stop Notification Request with the status code 0x100F.
R3-08	Invalid M(s)	Verify that when the SUT receives a Registration Response message with an invalid Message Sent field, it issues a Stop Notification Request with the status code 0x1014.
R3-09	Invalid M(r)	Verify that when the SUT receives a Registration Response message with an invalid Message Received field, it issues a Stop Notification Request with the status code 0x1015.
R3-10	Short Message	Verify that when the SUT receives the start Registration Response message but never receives the complete message, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
R3-11	Long Message	Verify that when the SUT receives a valid Registration Response message that has additional bytes immediately following it, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
R3-12	Smaller Message Length for Message Type	Verify that when the SUT receives a Registration Response message whose Message Length matches the bytes received, but is the wrong (smaller) length for this message type, that it issues a Stop Notification with the status code 0x1008, 0x100C, or 0x1019.
R3-13	Longer Message Length for Message Type	Verify that when the SUT receives a Registration Response message whose Message Length matches the bytes received, but is the wrong (larger) length for this message type, that it issues a Stop Notification with the status code 0x1008 or 0x100C.
New Tests introduced for v1.3 and Beyond		
R3-14	Undefined Flags	Verify that when the SUT receives a Registration Response message that has undefined Flag bits set for the version of CMHP, that it issues a Stop Notification with the status code 0x101B.

Test #	Overview	Test Purpose
R3-15	Invalid Source Location	**Optional Test for v1.3:** Verify that when the SUT receives a Registration Response message whose Source Location does not match the registered Source Location, it issues a Stop Notification with the status code 0x101C.
R3-16	Invalid Spare Field1	Verify that when the SUT receives a Registration Response message whose first Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101D.
R3-17	Invalid Spare Field2	Verify that when the SUT receives a Registration Response message whose second Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101E.
R3-18	Registration Response received with Poll flag set	Verify that when the SUT receives a Registration Response with the Poll flag set, it sends a Stop Notification with status code 0x1020.
R3-19	Registration Response received with Final flag set	Verify that when the SUT receives a Registration Response with the Final flag set, it sends a Stop Notification with status code 0x1021.

Additional Notes:

- Test R3-01 should also be used to verify the SUT's ability to handle the Partial Read Timeout and the accuracy of that timer.
- Test R3-15 is an optional test (for v1.3) that is run only if the SUT pre-defines the Source Location being used for the CMHP server (that is, CTE test tool). If, however, the SUT uses the Registration Response message to dynamically set the Source Location field for the CMHP server, then this test is not valid.

5.6.3.4 R4 – Registration Request – General Tests (Server)

The intent of this Test Group is to ensure that the SUT (when configured as a CMHP server) successfully handles a variety of incoming Registration Request messages for three different combinations of primary and secondary identifiers (identified below as USER1, USER2, and USER3):

- USER1 is used to indicate that the CMHP server (SUT) will expect the CMHP server (CTE test tool) to use only the PID field to register.
- USER2 is used to indicate that the CMHP server (SUT) will expect the CMHP Server (CTE test tool) to use both the PID and SID fields to register.
- USER3 is used to indicate that the CMHP server will define a user account that requires the CMHP client (CTE test tool) to use the maximum number of characters for the PID and SID fields.

Test #	Overview	Test Purpose
R4-01	Valid Primary Id only	Verify that when the SUT receives a valid Registration Request containing only a Primary Identifier (USER1), it sends a Registration Response with a success code in acknowledgment.
R4-02	Invalid Primary Id	Verify that when the SUT receives an invalid Primary Identifier (USER1), it sends a Registration Response with the error status 0x1001.
R4-03	Invalid Secondary Id	Verify that when the SUT receives a valid Primary Identifier, but receives an unexpected Secondary Identifier (USER1), it sends a Registration Response with the error status 0x1002.
R4-04	Valid Primary and Secondary Id	Verify that when the SUT receives a Registration Request containing valid Primary and Secondary Identifiers (USER2), it sends a Registration Response with a success code in acknowledgment.
R4-05	Valid Primary, invalid secondary identifier	Verify that when the SUT receives a Registration Request containing a valid Primary Identifier but an invalid Secondary Identifier (USER2), it sends a Registration Response with the error status 0x1002.
R4-06	Valid Primary Id, but no Secondary Id	Verify that when the SUT receives a Registration Request containing a valid Primary Identifier but fails to send a Secondary Identifier (USER2), it sends a Registration Response with the error status 0x1002.
R4-07	Access Barred	**Optional Test:** Verify that when the SUT disables a user account, subsequent attempts to register are blocked and the SUT sends a Registration Response with the error status 0x1003.
R4-08	Valid Primary and Secondary Id	Verify that when the SUT receives a Registration Request containing valid Primary and Secondary Identifiers (USER3) that use the maximum number of characters for the PID and SID fields, it sends a Registration Response with a success code in acknowledgment.
R4-09	Registration Request timeout	Verify that the SUT sends a Stop Notification with the status code (0x1010) when no Registration Request is received by the SUT within a timeout period after a socket is established.
R4-10	Registration Request timeout; Send SNR in response	Ensure that the SUT ignores the receipt of a Stop Notification Response after sending a Stop Notification Request indicating a Registration timeout.
R4-11	Registration Request received after a Registration Request timeout	Ensure that the SUT ignores a valid incoming Registration Request after sending a Stop Notification indicating a Registration timeout.
New Tests added for v1.2 and Beyond		
R4-12	Stop Notification Request	**Optional Test:** Only use when CMHP server (SUT) initiates socket connection to a CMHP client. A Stop Notification Request with a system-specific status code is sent instead of a Registration Request.

Additional Notes:

- Test R4-07 is an optional test for those SUTs that have the ability to disable user accounts when one or more failed registration attempts are detected.
- Test R4-09 should also be used to verify the accuracy of the SUT's Registration Timer.
- Test R4-12 is an optional test that is only valid for a SUT (CMHP server) that can initiate a TCP socket connection to the CTE test tool (CMHP client), and under CMHP v1.2 (and beyond), that the CMHP client can respond to the incoming socket request with a Stop Notification Request indicating that it is not ready to establish an application-level connection with the CMHP server. (If the CMHP client was ready, then the normal response to the incoming socket connection would be to send a Registration Request.)

5.6.3.5 R5 – Registration Request – Illegal Message Tests (Server)

The intent of this Test Group is to ensure that the SUT successfully handles the set of conditions where the CMHP client has initiated the TCP socket, but instead of sending in a Registration Request, it sends in another (valid) CMHP message type. In all cases, the SUT must issue a Stop Notification Request message with an appropriate status code.

Test #	Overview	Test Purpose
R5-01	Registration Response	Verify that when the SUT receives a valid Registration Response message, it issues a Stop Notification Request with the status code 0x100D, 0x100E, or 0x1013.
R5-02	Application data	Verify that when the SUT receives a valid application data message, it issues a Stop Notification Request with the status code 0x100D, 0x100E, or 0x1013.
R5-03	Acknowledgment	Verify that when the SUT receives a valid Acknowledgment message, it issues a Stop Notification Request with the status code 0x100D, 0x100E, or 0x1013.
R5-04	Stop Notification Request	Verify that when the SUT receives a valid Stop Notification Request message, it issues a Stop Notification Request with the status code 0x100D, 0x100E, or 0x1013.
R5-05	Stop Notification Response	Verify that when the SUT receives a valid Stop Notification Response message, it issues a Stop Notification Request with the status code 0x100D, 0x100E, or 0x1013.

5.6.3.6 R6 – Registration Request – Corrupted Message Tests (Server)

The intent of this Test Group is to ensure that the SUT successfully handles the set of conditions where it receives a corrupted Registration Request message. In all cases, the SUT shall issue a Stop Notification Request message with the appropriate status code.

Test #	Overview	Test Purpose
R6-01	Short Message Length	Verify that when the SUT receives a Registration Request message with a message length less than 40 bytes, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
R6-02	Long Message Length	Verify that when the SUT receives a Registration Request with a message length exceeding the number of bytes sent, it issues a Stop Notification Request with either the status code 0x1008, 0x100C, or 0x1019.
R6-03	Invalid Message Type	Verify that when the SUT receives a Registration Request message but the Message Type field contains an invalid value, it issues a Stop Notification Request with the status code 0x1009 or 0x101A.
R6-04	Invalid Major Id	Verify that when the SUT receives a Registration Request message with an invalid Major Id field, it issues a Stop Notification Request with the status code 0x100A.
R6-05	Invalid Minor Id	Verify that when the SUT receives a Registration Request message with an invalid Minor Id field, it issues a Stop Notification Request with the status code 0x100A.
R6-06	Invalid Checksum	Verify that when the SUT receives a Registration Request message with an invalid CRC field, it issues a Stop Notification Request with the status code 0x100F.
R6-07	Invalid M(s)	Verify that when the SUT receives a Registration Request message with an invalid Message Sent field, it issues a Stop Notification Request with the status code 0x1014.
R6-08	Invalid M(r)	Verify that when the SUT receives a Registration Request message with an invalid Message Received field, it issues a Stop Notification Request with the status code 0x1015.
R6-09	Short Message	Verify that when the SUT receives the start Registration Request message but never receives the complete message, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
R6-10	Long Message	Verify that when the SUT receives a valid Registration Request message that has additional bytes immediately following it, the SUT issues a Stop Notification Request with the status code 0x1008, 0x100A, 0x100C, or 0x1019.
R6-11	Short Message Length for Message Type	Verify that when the SUT receives a Registration Request message whose Message Length matches the bytes received but is the wrong (shorter) length for this message type, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
R6-12	Medium Message Length for Message Type	Verify that when the SUT receives a Registration Request message whose Message Length matches the bytes received but is the wrong (larger than CMHP Header but less than PID field) length for this message type, it issues a Stop Notification with the status code 0x1008, 0x100C, or 0x100D.
R6-13	Longer Message Length for Message Type	Verify that when the SUT receives a Registration Request message whose Message Length matches the bytes re but is the wrong (larger) length for this message type, it issues a Stop Notification with the status code 0x1008, 0x100C, or 0x100D.
New Tests introduced for v1.3 and Beyond		
R6-14	Undefined Flags	Verify that when the SUT receives a Registration Request message that has undefined Flag bits set for the version of CMHP, it issues a Stop Notification with the status code 0x101B.

Test #	Overview	Test Purpose
R6-15	Invalid Source Location	**Optional Test for v1.3:** Verify that when the SUT receives a Registration Request message whose Source Location does not match the registered Source Location, it issues a Stop Notification with the status code 0x101C.
R6-16	Invalid Spare Field 1	Verify that when the SUT receives a Registration Request message whose first Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101D.
R6-17	Invalid Spare Field 2	Verify that when the SUT receives a Registration Request message whose second Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101E.
R6-18	Poll flag set	Verify that when the SUT receives a Registration Request with the Poll flag set, it sends a Stop Notification with status code 0x1020.
R6-19	Final flag set	Verify that when the SUT receives a Registration Request with the Final flag set, it sends a Stop Notification with status code 0x1021.
R6-20	Non-zero Status field	Verify that when the SUT receives a Registration Request message whose Status Field is non-zero, it issues a Stop Notification with the status code 0x101F.

Additional Notes:

- Test R6-15 is an optional test that is required to be run only if the SUT pre-defines the Source Location for the CMHP client. However, if the SUT uses the incoming Registration Request message to dynamically set the Source Location field for the CMHP client, then this test is not valid.

5.6.3.7 R7 – Registration Violation – General Tests (Either)

The purpose of these tests is to send Registration Request messages after a successful registration has already occurred. These tests can be run against the SUT regardless of whether it is configured as a CMHP client or server application.

Test #	Overview	Test Purpose
R7-01	Same User	Verify that when the SUT receives a second Registration Request message from the same user, it issues a Stop Notification Request with the status code 0x100E or 0x1013.
R7-02	Different User	Verify that when the SUT receives a second Registration Request but this time for a different user, it issues a Stop Notification Request with the status code 0x100E or 0x1013.

5.6.3.8 R8 – Registration Violation – Corrupted Message Tests (Either)

The purpose of these tests is to send corrupted Registration Request messages after a successful registration has already occurred. These tests can be run against the SUT regardless of whether it is configured as a CMHP client or server application.

Test #	Overview	Test Purpose
R8-01	Short Message Length	Verify that when the SUT receives a second Registration Request message but with a message length less than 40 bytes, it issues a Stop Notification Request with the status code 0x1008, 0x100C, 0x100E, or 0x1019.
R8-02	Long Message Length	Verify that when the SUT receives a second Registration Request but with a message length exceeding the number of bytes sent, it issues a Stop Notification Request with either the status code 0x1008, 0x100C, 0x100E, or 0x1019.
R8-03	Invalid Message Type	Verify that when the SUT receives a second Registration Request message but the Message Type field contains an invalid value, it issues a Stop Notification Request with the status code 0x1009 or 0x101A.
R8-04	Invalid Major Id	Verify that when the SUT receives a second Registration Request message with an invalid Major Id field, it issues a Stop Notification Request with the status code 0x100E or 0x100A.
R8-05	Invalid Minor Id	Verify that when the SUT receives a second Registration Request message with an invalid Minor Id field, it issues a Stop Notification Request with the status code 0x100E or 0x100A.
R8-06	Invalid Checksum	Verify that when the SUT receives a second Registration Request message with an invalid CRC field, it issues a Stop Notification Request with the status code 0x100E or 0x100F.
R8-07	Invalid M(s)	Verify that when the SUT receives a second Registration Request message with an invalid Message Sent field, it issues a Stop Notification Request with the status code 0x100E, 0x1013 or 0x1014.
R8-08	Invalid M(r)	Verify that when the SUT receives a second Registration Request message with an invalid Message Received field, it issues a Stop Notification Request with the status code 0x100E, 0x1013, or 0x1015.
R8-09	Short Message	Verify that when the SUT receives the start of the second Registration Request message but never receives the complete message, it issues a Stop Notification Request with the status code 0x1008, 0x100C, 0x100E, 0x1013, or 0x1019.
R8-10	Long Message	Verify that when the SUT receives a second Registration Request message that has additional bytes immediately following it, it issues a Stop Notification Request with the status code 0x100E or 0x1013.
R8-11	Short Message Length for Message Type	Verify that when the SUT receives a second Registration Request message whose Message Length matches the bytes received but is the wrong (shorter) length for this message type, it issues a Stop Notification with the status code 0x1008, 0x100C, 0x100E, or 0x1019.
R8-12	Medium Message Length for Message Type	Verify that when the SUT receives a second Registration Request message whose Message Length matches the bytes received but is the wrong (larger than CMHP Header but less than PID field) length for this message type, it issues a Stop Notification with the status code 0x1008, 0x100C, 0x100D, 0x100E, 0x1013, or 0x1019.
R8-13	Longer Message Length for Message Type	Verify that when the SUT receives a second Registration Request message whose Message Length matches the bytes received but is the wrong (larger) length for this message type, it issues a Stop Notification with the status code 0x1008, 0x100C, 0x100E, 0x1013, or 0x100D.

5.6.3.9 A1 – Acknowledgments – General Tests (Either)

The purpose of this test group is to ensure that the SUT is able to handle the use of the Acknowledgment message, such as when it is used for the keep-alive mechanism, and to ensure that the SUT appropriately processes Acknowledgment messages.

Tests A1-01 and A1-02 require the CTE test tool to have its Keep-Alive Timer less than the Keep-Alive Timer of the SUT. The remaining tests require the CTE test tool to have a Keep-Alive Timer greater than the Keep-Alive Timer of the SUT.

Test #	Overview	Test Purpose
A1-01	Send Single Ack	Verify that when the SUT responds to the receipt of an Ack message (with the Poll bit set), it responds back with an Ack message (with the Final bit set).
A1-02	Send Multiple Acks	As above but verifies that the SUT keeps responding to Ack messages (Keep Alives) sent by the CTE test tool.
A1-03	Receive Multiple Acks	Reverse the timer relationship, and verify that the SUT generates Ack messages with the Poll bit set (Keep Alive) in accordance with its Keep-Alive Timer.
A1-04	Ack Timeout	Verify that the SUT sends a Stop Notification Request with the status of 0x1006 when it fails to receive an Ack from the CTE test tool to any of its Keep-Alive Acks.
A1-05	Ack Poll Bit Clear	Verify that the SUT ignores the receipt of an Ack with the Poll bit clear and initiates its own Keep-Alive mechanism.

Additional Notes:

- Test A1-03 is the first test that can be used to verify the accuracy of the SUT's Keep-Alive Timer.

5.6.3.10 A2 – Acknowledgments – Corrupted Message Tests (Either)

The purpose of these tests is to ensure that the SUT can handle "corrupted" Acknowledgment messages and initiates socket-closing procedures by issuing the Stop Notification Request with the appropriate status code for the condition detected.

Test #	Overview	Test Purpose
A2-01	Short Message Length	Verify that when the SUT receives an Acknowledgment message with a message length less than 40 bytes, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
A2-02	Long Message Length	Verify that when the SUT receives an Acknowledgment message with a message length exceeding the number of bytes sent, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
A2-03	Invalid Message Type	Verify that when the SUT receives an Acknowledgment message but the Message Type field contains an invalid value, it issues a Stop Notification Request with the status code 0x1009 or 0x101A.
A2-04	Invalid Major Id	Verify that when the SUT receives an Acknowledgment message with an invalid Major Id field, it issues a Stop Notification Request with the status code 0x100A.
A2-05	Invalid Minor Id	Verify that when the SUT receives an Acknowledgment message with an invalid Minor Id field, it issues a Stop Notification Request with the status code 0x100A.
A2-06	Invalid Checksum	Verify that when the SUT receives an Acknowledgment message with an invalid CRC field, it issues a Stop Notification Request with the status code 0x100F.
A2-07	Invalid M(s)	Verify that when the SUT receives an Acknowledgment message with an invalid Message Sent field, it issues a Stop Notification Request with the status code 0x1014.
A2-08	Invalid M(r)	Verify that when the SUT receives an Acknowledgment message with an invalid Message Received field, it issues a Stop Notification Request with the status code 0x1015.
A2-09	Short Message	Verify that when the SUT receives the start of an Acknowledgment message but never receives the complete message, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
A2-10	Long Message	Verify that when the SUT receives a valid Acknowledgment message that has additional bytes immediately following it, it processes the Ack, but then issues a Stop Notification Request with the status code 0x1008, 0x100A, 0x100C, or 0x1019 when the additional bytes are 0xA5.
A2-11	Smaller Message Length for Message Type	Verify that when the SUT receives an Acknowledgment message whose Message Length matches the bytes received but is the wrong (smaller) length for this message type, it issues a Stop Notification with the status code 0x1008, 0x100C, or 0x1019.
A2-12	Longer Message Length for Message Type	Verify that when the SUT receives an Acknowledgment message whose Message Length matches the bytes received but is the wrong (larger) length for this message type, it issues a Stop Notification with the status code 0x1008 or 0x100C.
New Tests added for v1.3 and Beyond		
A2-13	Undefined Flags	Verify that when the SUT receives an Acknowledgment message that has undefined Flag bits set for the version of CMHP, it issues a Stop Notification with the status code 0x101B.
A2-14	Invalid Source Location	Verify that when the SUT receives an Acknowledgment message whose Source Location does not match the registered Source Location for the Conformance Test system, it issues a Stop Notification with the status code 0x101C.

Test #	Overview	Test Purpose
A2-15	Invalid Spare Field 1	Verify that when the SUT receives an Acknowledgment message whose first Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101D.
A2-16	Invalid Spare Field 2	Verify that when the SUT receives an Acknowledgment message whose second Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101E.
A2-17	Non-zero Status Field	Verify that when the SUT receives an Acknowledgment message whose Status Field is non-zero, it issues a Stop Notification with the status code 0x101F.

5.6.3.11 S1 – Stop Notification Request – Corrupted Message Tests (Either)

The purpose of these tests is to ensure that the SUT can handle corrupted Stop Notification Request messages received in response to a SUT-initiated Stop Notification Request.

Test #	Overview	Test Purpose
S1-01	Short Message Length	Verify that when the SUT receives a Stop Notification Request message with a message length less than 40 bytes, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
S1-02	Long Message Length	Verify that when the SUT receives a Stop Notification Request with a message length exceeding the number of bytes sent, it issues a Stop Notification Request with either the status code 0x1008, 0x100C, or 0x1019.
S1-03	Invalid Message Type	Verify that when the SUT receives a Stop Notification Request message but the Message Type field contains an invalid value, it issues a Stop Notification Request with the status code 0x1009 or 0x101A.
S1-04	Invalid Major Id	Verify that when the SUT receives a Stop Notification Request message with an invalid Major Id field, it issues a Stop Notification Request with the status code 0x100A.
S1-05	Invalid Minor Id	Verify that when the SUT receives a Stop Notification Request message with an invalid Minor Id field, it issues a Stop Notification Request with the status code 0x100A.
S1-06	Invalid Checksum	Verify that when the SUT receives a Stop Notification Request message with an invalid CRC field, it issues a Stop Notification Request with the status code 0x100F.
S1-07	Invalid M(s)	Verify that when the SUT receives a Stop Notification Request message with an invalid Message Sent field, it issues a Stop Notification Request with the status code 0x1014.
S1-08	Invalid M(r)	Verify that when the SUT receives a Stop Notification Request message with an invalid Message Received field, it issues a Stop Notification Request with the status code 0x1015.
S1-09	Short Message	Verify that when the SUT receives the start of a Stop Notification Request message (less than 40 bytes) but never receives the complete message, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
S1-10	Long Message	Verify that when the SUT receives a valid 40-byte Stop Notification Request message that has additional bytes immediately following, it issues a Stop Notification Response, ignores the extra bytes, and shuts down.
S1-11	Smaller Message Length for Message Type	Verify that when the SUT receives a Stop Notification Request message whose Message Length matches the bytes received but is the wrong (smaller) length for this message type, it issues a Stop Notification with the status code 0x1008, 0x100C, or 0x1019.
S1-12	Longer Message Length for Message Type	Verify that when the SUT receives a Stop Notification Request message whose Message Length matches the bytes received but is the wrong (>256) length for this message type, it issues a Stop Notification with the status code 0x1008 or 0x100C.
New Tests added for v1.3 and Beyond		
S1-13	Undefined Flags	Verify that when the SUT receives a Stop Notification Request message that has undefined Flag bits set for the version of CMHP, it issues a Stop Notification with the status code 0x101B.
S1-14	Invalid Source Location	Verify that when the SUT receives a Stop Notification Request message whose Source Location does not match the registered Source Location, it issues a Stop Notification with the status code 0x101C.
S1-15	Invalid Spare Field 1	Verify that when the SUT receives a Stop Notification Request message whose first Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101D.

Test #	Overview	Test Purpose
S1-16	Invalid Spare Field 2	Verify that when the SUT receives a Stop Notification Request message whose second Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101E.
S1-17	Invalid Status	Verify that when the SUT receives a Stop Notification Request message that contains zero in the status field, it issues a Stop Notification with the status code 0x101F.

5.6.3.12 S2 – Stop Notification Request - Illegal Message Tests (Either)

The purpose of these tests is to verify that the SUT handles valid CMHP messages that are not valid for the current protocol state. Depending on its implementation, the SUT can either ignore all incoming messages or it will process all of them accordingly.

Note: For each of these tests, the SUT is required to be able to send a Stop Notification Request with a normal status code (<0x1000) on request to initiate the test. If not, then this group of tests is not valid.

Test #	Overview	Test Purpose
S2-01	Send Registration Request	Verify that when the SUT sends a Stop Notification Request Message and receives a Registration Request message in return, it ignores it (or sends a Stop Notification Request message with a status code of 0x100E or 0x1013) and closes the socket.
S2-02	Send Registration Response	Verify that when the SUT sends a Stop Notification Request Message and receives a Registration Response message in return, it ignores it (or sends a Stop Notification Request message with a status code of 0x100E or 0x1013) and closes the socket.
S2-03	Send application data message	Verify that when the SUT sends a Stop Notification Request Message and receives an Acknowledgment message in return, it ignores it (or responds with an Ack) and closes the socket.
S2-04	Send Ack with PB Clear	Verify that when the SUT sends a Stop Notification Request Message and receives an Acknowledgment message (with the Poll bit clear) in return, it ignores it and closes the socket.
S2-05	Send Ack with PB Set	Verify that when the SUT sends a Stop Notification Request Message and receives an Acknowledgment message (with the Poll bit set) in return, it ignores it (or responds with an Ack with the Final bit set) and closes the socket.
S2-06	Send Stop Notification Request	Verify that when the SUT sends a Stop Notification Request Message and receives a Stop Notification Request (with a normal status) in return, it sends a Stop Notification Response and then closes the socket.
S2-07	Timeout	Verify that when the SUT sends a Stop Notification Request Message and fails to receive a Stop Notification Response within a timeout period, it closes the socket.
S2-08	Normal Condition	Verify that the normal condition to respond to the Stop Notification Request Message with a Stop Notification Response is handled correctly.

Additional Notes:

- Test S2-08 is actually a valid exchange for a Stop Notification Request issued by the SUT, and has been added to this group (even though the rest of these tests are verification of illegal message conditions) solely based on the fact that the S2 and S3 tests are the only groups of tests where the Stop Notification Request is initiated by the SUT.

5.6.3.13 S3 – Stop Notification Response - Corrupted Message Tests (Either)

The purpose of these tests is to ensure that the SUT can handle corrupted Stop Notification responses received in response to a SUT-initiated Stop Notification Request. Depending on its implementation, the SUT can either ignore all incoming messages or it will process all of them accordingly.

Note: For each of these tests, the SUT is required to be able to send a Stop Notification Request with a normal status code (<0x1000) on request to initiate the test. If not, then this group of tests is not valid.

Test #	Overview	Test Purpose
S3-01	Short Message Length	Verify that when the SUT receives a Stop Notification Response with a message length less than 40 bytes, it ignores the message (or sends a Stop Notification Request with a status code of 0x1008, 0x100C, or 0x1019) and shuts down.
S3-02	Long Message Length	Verify that when the SUT receives a Stop Notification Response with a message length exceeding the number of bytes sent, it ignores the message (or sends a Stop Notification Request with a status code of 0x1008, 0x100C, or 0x1019) and shuts down.
S3-03	Invalid Message Type	Verify that when the SUT receives a Stop Notification Response message but the Message Type field contains an invalid value, it ignores the message (or sends a Stop Notification Request with a status code of 0x1009 or 0x101A) and shuts down.
S3-04	Invalid Major Id	Verify that when the SUT receives a Stop Notification Response message with an invalid Major Id field, it ignores the message (or sends a Stop Notification Request with a status code of 0x100A) and shuts down.
S3-05	Invalid Minor Id	Verify that when the SUT receives a Stop Notification Response message with an invalid Minor Id field, it ignores the message (or sends a Stop Notification Request with a status code of 0x100A) and shuts down.
S3-06	Invalid Checksum	Verify that when the SUT receives a Stop Notification Response message with an invalid CRC field, it ignores the message (or sends a Stop Notification Request with a status code of 0x100F) and shuts down.
S3-07	Invalid M(s)	Verify that when the SUT receives a Stop Notification Response with an invalid Message Sent field, it ignores the message (or sends a Stop Notification Request with a status code of 0x1014) and shuts down.
S3-08	Invalid M(r)	Verify that when the SUT receives a Stop Notification Response message with an invalid Message Received field, it ignores the message (or sends a Stop Notification Request with a status code of 0x1015) and shuts down.
S3-09	Short Message	Verify that when the SUT receives the start of a Stop Notification Request message but never receives the complete message, it ignores the partial message (or sends a Stop Notification Request with a status code of 0x1008, 0x100C, or 0x1019) and shuts down.
S3-10	Long Message	Verify that when the SUT receives a Stop Notification Response that has additional bytes immediately following it, it issues a Stop Notification Response, ignores the subsequent bytes (or sends a Stop Notification Request with a status code of 0x1008, 0x100C, or 0x1019), and shuts down.
S3-11	Smaller Message Length for Message Type	Verify that when the SUT receives a Stop Notification Response message whose Message Length matches the bytes received but is the wrong (smaller) length for this message type, it ignores the message (or sends a Stop Notification Request with a status code of 0x1008, 0x100C, or 0x1019) and shuts down.
S3-12	Longer Message Length for Message Type	Verify that when the SUT receives a Stop Notification Response message whose Message Length matches the bytes received but is the wrong (larger) length for this message type, that it ignores the message (or sends a Stop Notification Request with a status code of 0x1008 or 0x100C) and shuts down.

Test #	Overview	Test Purpose
New Tests added for v1.3 and Beyond		
S3-13	Undefined Flags	Verify that when the SUT receives a Stop Notification Response message that has undefined Flag bits set for the version of CMHP, that it issues a Stop Notification with the status code 0x101B.
S3-14	Invalid Source Location	Verify that when the SUT receives a Stop Notification Response message whose Source Location does not match the registered Source Location, that it issues a Stop Notification with the status code 0x101C.
S3-15	Invalid Spare Field 1	Verify that when the SUT receives a Stop Notification Response message whose first Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101D.
S3-16	Invalid Spare Field 2	Verify that when the SUT receives a Stop Notification Response message whose second Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101E.

5.6.3.14 S4 – Stop Notification Response - Illegal Message Tests (Either)

The purpose of this test is to ensure that the SUT can handle a Stop Notification response received after a successful registration.

Test #	Overview	Test Purpose
S4-01	Send Stop Notification Response	Verify that when the SUT receives a Stop Notification Response after a successful registration, it issues a Stop Notification Request with a status code of 0x100E or 0x1013, and then shuts down.

5.6.3.15 D1 – Application Data Transfer - Corrupted Message Tests (Either)

The purpose of these tests is to ensure that the SUT can handle corrupted application data messages.

Test #	Overview	Test Purpose
D1-01	Short Message Length	Verify that when the SUT receives a data message with a corrupted message length, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
D1-02	Long Message Length	Verify that when the SUT receives a data message with a message length exceeding the number of bytes sent, it issues a Stop Notification Request with either the status code 0x1008, 0x100C, or 0x1019.
D1-03	Invalid Message Type	Verify that when the SUT receives a data message but the Message Type field contains an invalid value, it issues a Stop Notification Request with the status code 0x1009 or 0x101A.
D1-04	Invalid Major Id	Verify that when the SUT receives a data message with an invalid Major Id field, it issues a Stop Notification Request with the status code 0x100A.
D1-05	Invalid Minor Id	Verify that when the SUT receives a data message with an invalid Minor Id field, it issues a Stop Notification Request with the status code 0x100A.
D1-06	Invalid Checksum	Verify that when the SUT receives a data message with an invalid CRC field, it issues a Stop Notification Request with the status code 0x100F.
D1-07	Invalid M(s)	Verify that when the SUT receives a data message with an invalid Message Sent field, it issues a Stop Notification Request with the status code 0x1014.
D1-08	Invalid M(r)	Verify that when the SUT receives a data message with an invalid Message Received field, it issues a Stop Notification Request with the status code 0x1015.
D1-09	Short Message	Verify that when the SUT receives the start of a data message but never receives the complete message, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
D1-10	Long Message	Verify that when the SUT receives a valid data message that has additional bytes immediately following it, it processes the data message but then issues a Stop Notification Request with the status code 0x1008, 0x100A, 0x100C, or 0x1019 when the additional bytes 0xA5 are processed.
D1-11	Smaller Message Length for Message Type	Verify that when the SUT receives a data message whose Message Length matches the bytes received but is the wrong (less than 40 bytes) length for this message type, it issues a Stop Notification Request with the status code 0x1008, 0x100C, or 0x1019.
D1-12	Minimum Message Length for Message Type	**Optional Test**: Verify that when the SUT receives a data message that is smaller than the minimum message length for the application, it issues a Stop Notification with the status code 0x1008, 0x100C, or 0x1019.
D1-13	Wrong data message type	Verify that when the SUT receives a data message containing an invalid message type for the SUT's application, the SUT sends a Stop Notification Request with the status code 0x1009 or 0x101A.

New Tests added for v1.3 and Beyond		
D1-14	Undefined Flags	Verify that when the SUT receives a data message that has undefined Flag bits set for the version of CMHP, it issues a Stop Notification with the status code 0x101B.
D1-15	Invalid Source Location	Verify that when the SUT receives a data message whose Source Location does not match the registered Source Location, it issues a Stop Notification with the status code 0x101C.
D1-16	Invalid Spare Field 1	Verify that when the SUT receives a data message whose first Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101D.
D1-17	Invalid Spare Field 2	Verify that when the SUT receives a data message whose second Spare Field is not all zeros, it issues a Stop Notification with the status code 0x101E.

Additional Test Note:

- Test D1-12 is only valid if the SUT checks for a minimum length on its application-level data messages.

5.6.3.16 D2 – Application Data Transfer - SUT Receiving Tests (Either)

The purpose of this group of tests is to increase the level of confidence that the SUT can receive application-level data messages and provide a timely acknowledgment to all of them. The final test verifies that the SUT can handle the rollover condition for the Message Received count.

Test #	Overview	Test Purpose
D2-01	Send 1 data message	Verify that when the SUT receives a data message, it responds with an Acknowledgment message in a timely manner.
D2-02	Send 2 data messages	Verify that when the SUT receives two data messages, it acknowledges both in a timely manner.
D2-03	Send 10 data messages	Verify that the SUT can receive 10 data messages and that it acknowledges all of them in a timely manner.
D2-04	Send 100 data messages	Verify that the SUT can receive 100 data messages and that it acknowledges all of them in a timely manner.
D2-05	Send 300 data messages	Verify that the SUT can receive 300 data messages and that it acknowledges all of them in a timely manner. The test ensures that the SUT's handling of the M(r) rollover is correct.

5.6.3.17 D3 – Application Data Transfer - SUT Sending Tests (Either)

The purpose of this group of tests is to increase the level of confidence that the SUT can send application-level data messages and process the associated acknowledgments from the CTE test tool. The final test verifies that the SUT can handle the rollover condition for the Message Sent count.

Test #	Overview	Test Purpose
D3-01	Send No Ack	Verify that if the SUT receives no acknowledgment to a sent message, it prompts the CTE test tool for an acknowledgment and sends a Stop Notification with a status code of 0x1006 when it reaches its maximum number of retries.
D3-02	Send Ack, but not with an updated M(r)	Verify that if the SUT receives no indication to a sent message, that it prompts the CTE test tool for an acknowledgment and sends a Stop Notification with a code of 0x1006 when it reaches its maximum number of retries. The CTE test tool sends two Acknowledgments messages back – the first after immediate receipt of the sent message (with no flags set and no updated M(r)) and later when prompted by the SUT – this time with the Final bit set, but again does not update M(r). These Acknowledgments messages should be ignored by the SUT and have no impact to its retry mechanism.
D3-03	Send ack with updated M(r)	Verify that the SUT prompts for a response after sending a message and after it receives an updated M(r) that the next Acknowledgment message it sends is part of its Keep-Alive mechanism.
D3-04	Send an updated M(r) with a Shutdown message	Verify that the SUT handles an updated M(r) as part of the incoming Shutdown Notification message.
D3-05	Receive two messages and ack	Verify that the SUT can send two messages and that it handles the associated acknowledgments.
D3-06	Receive 10 messages and ack	Verify that the SUT can send 10 messages and that it handles the associated acknowledgments.
D3-07	Receive 20 messages and ack	Verify that the SUT can send 20 messages and that it handles the associated acknowledgments.
D3-08	Receive 100 messages and ack	Verify that the SUT can send 100 messages and that it handles the associated acknowledgments.
D3-09	Receive 300 messages and ack	Verify that the SUT can send 300 messages and that it handles the associated acknowledgments. Verifies that the SUT handles the M(s) rollover correctly.
D3-10	Verify SUT's Transmit Window	**Optional Test:** Verify that the SUT transmits its Tx window size before it requests an acknowledgment from the CTE test tool.

Additional Test Notes:

- Test D3-01 should also be used to verify the SUT's ability to use the Poll Response Timer and verify its accuracy.
- Test D3-03 should be used to verify the SUT's ability to switch between using its Poll Response Timer to its Keep-Alive Timer.
- Test D3-10 is an optional test and is only applicable where the SUT's Transmit Window Size is greater than 1.

5.6.3.18 F1 – Flow Control Tests (Server – v1.2 and Beyond)

The purpose of this group of tests is to verify that a CMHP server is able to handle the basic flow control mechanism.

Test #	Overview	Test Purpose
F1-01	Flow control stated at Registration	Verifies that a CMHP server (with messages ready to send) accepts an incoming Registration Request Message with the Flow Control flag set, and then responds to subsequent periodic Keep-Alive messages that also have the Flow Control flag, but never sends the queued messages.
F1-02	Flow control stated at Registration and incoming data received. Outgoing data sent after flow control lifted	Verifies that a CMHP server (with messages to send) accepts an incoming Registration Request Message with the Flow Control flag set. Verify that the SUT receives incoming messages with the Flow Control flag set and acknowledges receipt accordingly. When the SUT receives an Acknowledgment message with the Flow Control flag cleared, it then transmits the queued messages and receives the appropriate acknowledgment.

Additional Note:

- For both of these tests, the SUT is required to have messages queued up ready to send.

5.6.3.19 F1 – Flow Control Tests (Client - v1.2 and Beyond)

The purpose of this group of tests is to verify that a CMHP server is able to handle the basic flow control mechanism.

Test #	Overview	Test Purpose
F2-01	Flow control stated on Registration Response	Verify that the SUT does not send its queued messages when it receives a Registration Response with the Flow Control flag set. Verify that the SUT does not send its queued messages until it receives an Acknowledgment message with the Flow Control bit cleared.
F2-02	Flow control stated on Registration Response – send data	Verify that the SUT does not send a queued data message when it receives a Registration Response with the Flow Control flag set. Verify that the SUT receives messages and only sends its messages when the Flow Control flag is cleared.

Additional Note:

- For both of these tests, the SUT is required to have messages queued up ready to send.

6 NOTES

6.1 Intended Use

This handbook is intended to provide guidance regarding the application-level protocol, CMHP. The document is intended for those individuals who will play a role in designing TCP/IP connections between systems that require a degree of assurance that the data sent has been successfully received and processed by the far-end application.

6.2 Superseding Documentation

This section is not applicable to this handbook at this time.

6.3 Cross-Reference of Classifications and Substitutability Data

This section is not applicable to this handbook at this time.

6.4 Subject Term (Key Word) Listing

CMHP, application-level protocol, message delivery assurance

6.5 International Interest

This section is not applicable to this handbook at this time.

6.6 Identification of Changes

This is the first issue of this handbook.

6.7 Updating this Handbook

This handbook is designed to be updated and refined over time through changes in the NAS. The change process is only possible if the users of this handbook comment on its use. Users should keep notes regarding the areas where the handbook is either helpful or where it is lacking. These notes and comments about the handbook should then be provided to the organization referenced in the Foreword.