

BINARY TRANSFER PROTOCOL

**HAL Communications Corp.
Engineering Document E2004 Rev B
December 17, 1997**

Acknowledgment

The binary file transfer protocol described in this HAL Engineering Document is based on work done in 1993 by Mr. Peter Schulze of EURAF in Benin, Africa as part of his "EXPRESS" terminal software system. The HAL binary file transfer protocol described in this document is a modified version created by Dr. Andrew White of ABW Associates. HAL wishes to thank Mr. Schulze and Dr. White for their efforts.

1.0 INTRODUCTION

This document describes the HAL Binary Transfer Protocol used to transparently send binary disk files from one CLOVER station to another. Note that these files may contain any data bytes; binary, ASCII text, image, or database files are all treated the same way. The protocol is designed such that no remote user intervention is required to receive a file. If, for any reason, the link fails before a file transfer is complete, the partially received file is stored. Once the link is re-established, the next file transfer operation will detect the partial file and only send the last part of the original file.

The binary transfer protocol operates very efficiently because the files are compressed using the PKWARE Data Compression Library IMplode() routine before transmission. Depending on the actual data contained in the file, the compression ratio may be as high as 5:1. Even though compression is available, the protocol allows the transmission of uncompressed files as well.

In the following sections the HAL CLOVER Binary Transfer Protocol is described. First, each of the commands is summarized then examples are provided for typical operations. Note that the HAL PCC.EXE terminal program continuously monitors the RX data stream searching for the protocol commands. Thus, there is no configuration option needed to enable this binary transfer capability.

2.0 COMMAND CODES

The command and response codes are listed in this section in numerical order. Each command begins with ASCII SOH (0x01) escape character followed by one or more argument bytes. Note that all of the numbers are shown in hex. If an ASCII SOH character is received but no valid argument byte follows, the SOH is ignored and no error is signaled.

The following explanations assume that two stations are ARQ linked, and one station is sending a file to the other station. Currently, PCC.EXE does not permit the simultaneous sending and receiving of a binary file.

2.1 Send 0x00 Byte

Code: (0x01)(0x90)

The file sending station uses this sequence to send an ASCII NULL byte (0x00) during a file transfer to avoid the PCI-4000 chat mode NULL problem. During chat mode, each 6 character chat CCB block is NULL filled when there are fewer than 6 characters to send.

2.2 Send 0x01 Byte

Code: (0x01)(0x91)

The file sending station uses this sequence to send an ASCII SOH (0x01) byte during a file transfer.

2.3 File Already Exists

Code: (0x01)(0x92)

This sequence is sent by the file receiving station to reject a file transfer request since the complete file already exists. The file receiving station determines that the file exists by comparing the stored file length to the compressed length reported in the file transfer request (see Section 2.9). If the disk file length is greater than the compressed length, this response sequence is transmitted to the file sending station, and the file transfer stops.

2.4 End of File

Code: (0x01)(0x93)

The file sending station transmits this sequence after the last file data byte is sent to signal the end of the file. The file receiving station will respond with either success or failure depending on the received file length (see Sections 2.5 and 2.6).

2.5 File Received Successfully

Code: (0x01)(0x94)

This sequence is sent by the file receiving station when a file is successfully received. After the file receiving station receives the End of File command (see Section 2.4) from the file sending station, the receive file is closed and the stored file length is compared to the compressed file length reported in the initial file transfer request (see Section 2.9). If the file lengths match, then the file is exploded, if required, using the compression method specified in the file transfer request. Then the uncompressed file length is compared to the length reported in the initial file transfer request. If the uncompressed file is the correct length, then the File Received Successfully sequence is sent by the file receiving station. If the lengths do not match, then a failure is reported (see Section 2.6).

2.6 File Transfer Failed

Code: (0x01)(0x95)

This sequence is sent by the file receiving station when the file transfer fails. A failure occurs if the compressed file, as received, is not equal to the length reported in the file transfer request (see Section 2.9), or if the exploded file length is not equal to the uncompressed file length.

2.7 Stop File Transfer

Code: (0x01)(0x96)

The file receiving station or the file sending station sends this command to stop a file transfer. Generally, this command will only be sent if a user manually stops the transfer. The file receiving station should save the partially received file on disk for later completion.

2.8 Request Version Information

Code: (0x01)(0x97)

This command requests the receiving station software Version information. The reply is sent as part of the normal text stream, and it should simply be shown in the RX buffer. PCC Release 6 or later will respond to this command with all software version numbers and the PCI-4000 I/O address.

2.9 Request File Transfer

Code: (0x01)(0x80)filename(0x08)file_size(0x08)compressed_size
(0x08)compression_method(0x02)

This command signals the start of a binary file transfer. The `compressed_size` value is used to determine if a partial file already exists on the disk at the file receiving station. In addition, the length of the file received is compared to the `compressed_size` after the file is received. The `file_size` value is used after the file has been received and exploded to confirm that the file transfer was successful. `file_size` and `compressed_size` are ASCII number strings reporting the decimal file length with no leading zeros and no comma separators.

If a file with the same name exists on the disk, the length of that file is compared to `compressed_size`. If the file is longer than `compressed_size`, then the file receiving station reports to the file sending station that the file already exists (see Section 2.3) and the file transfer stops. If the file length is less than the `compressed_size`, then the file receiving station sends the stored file length to the file sending station; this is where the file sending station will start sending the compressed file (see Section 2.10).

The `compression_method` is an ASCII string. Currently, only two compression methods are recognized: PKLIB and NONE. PKLIB uses the PKWARE Data Compression Library `IMPLODE()` and `EXPLODE()` routines to compress and expand disk files (see Appendix A). Note that file compression occurs at the file sending station before the file transfer request is sent so that the compressed file length can be included. If NONE is selected, the `compressed_size` and `file_size` numbers will be equal.

2.10 Number of Bytes Received

Code: (0x01)(0x81)number_of_bytes_already_received(0x02)

This sequence is sent by the file receiving station in response to a file transfer request (see Section 2.9) to indicate the number of bytes already received during a previous transmission, if any. The number is an ASCII string reporting the decimal file length with no leading zeros and no comma separators. If the file does not exist on the disk, a file length of "0" is returned. If the Clover link fails during a transfer, the partial file is saved, and only the end of the file needs be sent the next time. Once this response is sent, the file receiving station is ready to receive the file from the file sending station.

If the file receiving station cannot expand the compression method indicated in the file transfer request, it responds with a list of the available compression methods (see Section 2.11) instead of this response.

2.11 Compression Methods Available

Code: (0x01)(0x82)method1(0x08)method2(0x02)

This sequence is sent by the file receiving station if the file sending station has selected a compression technique that the file receiving station cannot expand. More than one compression method may be announced with this command. The information sending station then begins a new transfer request using one of the available methods or NONE (see Section 2.9).

2.12 Chat Mode Characters

Code: (0x01)(0x83)keyboard_chat_characters(0x02)

This sequence is sent by the file sending station during a file transfer to send keyboard data in the RX buffer. The characters contained in this protocol message are not stored in the binary file. Using this command, a keyboard to keyboard conversation may be conducted while the file transfer is running in the background. There is no limit to the number of characters that this message may contain.

3.0 FILE TRANSFER EXAMPLES

Listed below are some typical exchanges. Note that these examples assume we are viewing the communications at the file sending station end of an ARQ link.

<-- represents data from the file sending station
--> represents data from the file receiving station

3.1 File Transfer

In this example, the file sending station requests a file transfer, the file receiving station responds, and the file is transferred.

```
<-- (0x01)(0x80)TEST.DAT(0x08)12345(0x08)1234(0x08)PKLIB(0x02)
--> (0x01)(0x81)0(0x02) - start sending at byte 0000
<-- DATA...DATA...DATA - file data
<-- (0x01)(0x83)CHAT..CHAT..CHAT(0x02) - keyboard chat data
<-- DATA...DATA...DATA - file data
<-- (0x01)(0x93) - signal end of file
--> (0x01)(0x94) - file received OK
```

3.2 File Already Exists

In the following example, the file already exists at the file receiving end, so the file transfer is terminated.

```
<-- (0x01)(0x80)TEST.DAT(0x08)12345(0x08)1234(0x08)PKLIB(0x02)
--> (0x01)(0x92) - file already exists
```

3.3 Partial File Exists

In this example, the file receiving station has a partial file, received during a previous transfer that was interrupted. The file sending station begins with the byte number indicated by the file receiving station response.

```
<-- (0x01)(0x80)TEST.DAT(0x08)12345(0x08)1234(0x08)PKLIB(0x02)
--> (0x01)(0x81)567(0x02) - start sending at byte number 567
<-- DATA...DATA...DATA - file data
<-- (0x01)(0x93) - signal end of file
--> (0x01)(0x94) - file received OK
```

3.4 File Transfer Fails

In this example, the file transfer fails because the compressed received file is not the length reported in the file transfer request command, or the uncompressed file is not the correct length.

```
<-- (0x01)(0x80)TEST.DAT(0x08)12345(0x08)1234(0x08)PKLIB(0x02)
--> (0x01)(0x81)123(0x02) - start sending at byte 123
<-- DATA...DATA...DATA - file data
<-- (0x01)(0x93) - signal end of file
--> (0x01)(0x95) - file transfer FAILED!
```

3.5 File Transfer Compression Mode Change

In the following example, the file receiving station does not know about the compression method selected in the file transfer request. The file receiving station responds with the compression methods it has, then it waits for another file transfer request.

```
<-- (0x01)(0x80)TEST.DAT(0x08)12345(0x08)2345(0x08)XYZ(0x02)
--> (0x01)(0x82)PKLIB(0x02) - use PKLIB compression
<-- (0x01)(0x80)TEST.DAT(0x08)12345(0x08)1234(0x08)PKLIB(0x02)
--> (0x01)(0x81)0(0x02) - start sending at byte 0000
<-- DATA...DATA...DATA - file data
<-- (0x01)(0x93) - signal end of file
--> (0x01)(0x94) - file received OK
```

3.6 File Transfer with No Compression

In the following example, the sending and receiving stations do not have a common compression method. As a result, the sending station sends the file uncompressed.

```
<-- (0x01)(0x80)TEST.DAT(0x08)12345(0x08)2345(0x08)XYZ(0x02)
--> (0x01)(0x82)PKLIB(0x02) - use PKLIB compression
<-- (0x01)(0x80)TEST.DAT(0x08)12345(0x08)12345(0x08)NONE(0x02)
--> (0x01)(0x81)0(0x02) - start sending at byte 0000
<-- DATA...DATA...DATA - file data
<-- (0x01)(0x93) - signal end of file
--> (0x01)(0x94) - file received OK
```

4.0 PROTOCOL LIMITATIONS

There are certain limitations to what the HAL Binary Transfer Protocol can do. If, for example, a different file happens to exist at the receiving station with the same name as the file name in the transfer request, unexpected results may occur. If this file has a length less than the compressed file length, the transfer will continue just as though a partially received file already existed. When file expansion occurs, the resulting file will, probably, not be the same length as the original file, and the transfer will fail. However, the pre-existing file will be destroyed.

In addition, the transfer protocol has no way to know what compression technique was used to store a partially received file. Should a second transfer request arrive with a different file compression technique, or NONE, then the file transfer will, probably, fail.

One final note for PCC.EXE users; we cannot send files with file name extensions ending in underbar "_". During file compression, a new file is created and the last character of the file extension is changed to this character. If one attempts to send a file with this name, the transfer will fail, and the original file may be deleted.

APPENDIX A - PKLIB ROUTINES

PKWARE, Inc, sells a product called the Data Compression Library that contains two linkable modules to perform file compression and expansion. Even though applications developers must purchase this product from PKWARE, the price includes an unlimited distribution license. Any program that you write using these routines may be freely distributed as long as the compression and expansion features are part of a larger application program.

For further information, please contact PKWARE directly:

PKWARE, Inc.
9025 N. Deerwood Drive
Brown Deer, WI 53223-2437

Voice: (414) 354-8699
FAX: (414) 354-8559
BBS: (414) 354-8670