CLARION TECHNICAL BULLETIN

Bulletin #120

File Format of Clarion Help Files

Overview

This bulletin contains details about the file format of Clarion Help files.

File Format of Clarion Help Files

This bulletin contains details about the file format of Clarion Help files, and assumes that you possess a basic understanding of C data structures. (For an overview of how help files work, see Clarion Technical Bulletin #112, "Clarion Help Files.")

As you may already know, parts of the .HLP files are compressed. It is important for you to know how this compression works in order to store and retrieve help window information outside of Helper or the Clarion environment. First, we will discuss the format of Clarion help files, and then take a look at the compression routine.

The following help file dump contains three help windows. The windows are named HELP1, HELP2, and HELP3. The HELP1 window chains to the HELP2 window, which chains to the HELP3 window. There are no menu fields in any of these windows, which will keep the explanation of the dump fairly simple.

```
0000: E0 49 AC 01 00 00 68 00 0D 00 06 00 0F 3E 06 0A |.I....h......>..|
0010: 01 CD 2F 48 45 4C 50 32 20 20 20 01 C9 00 CD 3C |../HELP2   ....<|
0020: BB BA 00 20 3C BA BA 00 20 3C BA BA 00 20 3C BA |... <... <... <.|
0030: BA 00 20 3C BA BA 00 20 3C BA BA 00 20 18 48 45 |.. <... <... .HE|
0040: 4C 50 20 57 49 4E 44 4F 57 20 31 00 20 17 BA BA |LP WINDOW 1. ...|
0050: 00 20 3C BA BA 00 20 3C BA BA 00 20 3C BA BA 00 |. <... <... <...|
0060: 20 3C BA BA 00 20 3C BA BA 00 20 3C BA BA 00 20 | <... <... <... |
0070: 3C BA C8 00 CD 3C BC 00 2F FF 00 2F FF 00 2F FF |<....<../../../.|
0080: 00 2F A5 01 00 00 FF 00 00 FF 00 00 FF 00 00 A5 |./..............|
0090: 06 0A 14 47 2F 01 68 00 0D 00 06 00 0F 3E 06 0A |...G/.h......>..|
00A0: 01 CD 2F 48 45 4C 50 33 20 20 20 01 C9 00 CD 3C |../HELP3   ....<|
00B0: BB BA 00 20 3C BA BA 00 20 3C BA BA 00 20 3C BA |... <... <... <.|
00C0: BA 00 20 3C BA BA 00 20 3C BA BA 00 20 18 48 45 |.. <... <... .HE|
00D0: 4C 50 20 57 49 4E 44 4F 57 20 32 00 20 17 BA BA |LP WINDOW 2. ...|
00E0: 00 20 3C BA BA 00 20 3C BA BA 00 20 3C BA BA 00 |. <... <... <...|
00F0: 20 3C BA BA 00 20 3C BA BA 00 20 3C BA BA 00 20 | <... <... <... |
0100: 3C BA C8 00 CD 3C BC 00 2F FF 00 2F FF 00 2F FF |<....<../../../.|
0110: 00 2F A5 01 00 00 FF 00 00 FF 00 00 FF 00 00 A5 |./..............|
0120: 06 0A 14 47 2F 01 68 00 0D 00 06 00 0F 3E 06 0A |...G/.h......>..|
0130: 00 01 C9 00 CD 3C BB BA 00 20 3C BA BA 00 20 3C |.....<... <... <|
0140: BA BA 00 20 3C BA BA 00 20 3C BA BA 00 20 3C BA |... <... <... <.|
0150: BA 00 20 18 48 45 4C 50 20 57 49 4E 44 4F 57 20 |.. .HELP WINDOW |
0160: 33 00 20 17 BA BA 00 20 3C BA BA 00 20 3C BA BA |3. .... <... <..|
0170: 00 20 3C BA BA 00 20 3C BA BA 00 20 3C BA BA 00 |. <... <... <...|
0180: 20 3C BA BA 00 20 3C BA C8 00 CD 3C BC 00 2F FF | <... <....<../.|
0190: 00 2F FF 00 2F FF 00 2F A5 01 00 00 FF 00 00 FF |./../../........|
01A0: 00 00 FF 00 00 A5 06 0A 14 47 2F 01 48 45 4C 50 |.........G/.HELP|
01B0: 31 20 20 20 06 00 00 00 48 45 4C 50 32 20 20 20 |1   ....HELP2   |
01C0: 96 00 00 00 48 45 4C 50 33 20 20 20 26 01 00 00 |....HELP3   &...|
```

We have given names to the different areas. Field names within those areas will be supplied to identify the data that is stored there. Preceding each field name is the offset in hexidecimal for this file, relative to zero. The important areas will be discussed in detail in the next section.

If you use Scanner to look at a help file, add one to each offset you see here since Scanner starts the file at offset 0001. We will use the HELP1 window to show how the windows are retrieved.

FILE HEADER:

```
struct   {
  unsigned signature;                    /* file signature                    */
  unsigned long windlist;                /* offset to window list             */
  };
```

(0000) signature — The file's signature (E0 49). This field tells us that this is a file created by Helper (chlp.exe) version 2. Since the bytes are stored in reverse order due to the Intel architecture, the value is actually hex 49 E0. If this value is something other than E0 49, version 2 of Helper will NOT recognize this as a help file.

(0002) windlist — The offset into the window list (AC 01 00 00). The window list contains the name(s) of the window(s) and the offset(s) into each window's header information. Again, since the bytes are stored in reverse order, and this is an unsigned long, the value is actually hex 000001AC.

WINDOW LIST STRUCTURE:

```
struct  {
  char windid[8];                        /* window name                       */
  unsigned long woffset;                 /* offset of window header           */
  };
```

(01AC) windid — The name of the first help window in the window list (HELP1).

(01B4) woffset — The offset of this window's header information (06 00 00 00).

After the first window list structure, you will find the structures for all of the other windows stored in this help file.

WINDOW HEADER:

```
struct {
  unsigned windbuf;                      /* size of window buffer             */
  unsigned contbuf;                      /* size of control buffer            */
  unsigned paintbuf;                     /* size of paint buffer              */
  char nol;                              /* number of lines in window         */
  char noc;                              /* number of columns in window       */
  char trow;                             /* beginning row of window           */
  char tcol;                             /* beginning column of window        */
  char chain;                            /* chain to window                   */
  };
```

(0006) windbuf — The length of the compressed window buffer (68 00). The window buffer is where the characters and their attributes are stored for the HELP1 window.

(0008) contbuf — The length of the compressed control buffer (0D 00). The control buffer will follow the window buffer. This buffer contains attributes that are used by Helper.

(000A) paintbuf — The length of the paint buffer (06 00). The paint buffer will follow the control buffer. This buffer contains paint attributes that are used by Helper.

(000C) nol — The number of lines in the window (0F).

(000D) noc — The number of columns in the window (3E).

(000E) trow — The beginning row of the window (06). If the high order bit of this byte is set to ON (i.e. 86h), then this window position would be FIXED at the row indicated. The actual row used to display a FLOATING help window in the Clarion environment is determined where it is being referenced.

(000F) tcol — The beginning column of the window (0A). Again, if the fixed indicator was on in the row field, this column will be the fixed location of the window. Floating windows determine the column where they are referenced.

(0010) chain — This byte indicates that there is a chain to another window from this window (01).

```
      00 -> no chain or menu
      01 -> chain to another window
>     01 -> menu items in this window
      menu items = chain - 1
```

As you can see, you cannot have both a chain and a menu item for a given window.

WINDOW CHAIN STRUCTURE:

```
struct              {
  unsigned overh;                   /* helper overhead                  */
  char chwindow[8];                 /* chain to window name             */
  };
```

(0011) overh — These two bytes are overhead for Helper (CD 2F).

(0013) chwindow — The name of the help window that this window chains to (HELP2).

(The areas below and the compression technique will be discussed in a later section.)

(001B) — This byte is where the compressed window buffer begins. The number of bytes to read for this buffer is in the windbuf variable in the window header structure.

(0083)	This byte is where the compressed control buffer begins. The number of bytes to read for this buffer is in the contbuf variable in the window header structure.
(0090)	This byte is where the paint buffer begins. This buffer is not compressed. The number of bytes to read for this buffer is in the paintbuf variable in the window header structure.

WINDOW BUFFER AREA

This area, when decompressed, contains the characters and attributes for the help window. The characters are stored in the first half of the buffer and the attributes corresponding to each character are stored in the second half of the buffer.

CONTROL BUFFER AREA

This area, when decompressed, contains the control character for each window character. The control character is used by Helper to determine whether to use the paint attribute or use the character attribute for that character. In our example, all the control characters are set to zero. This tells Helper to use the paint attributes in the paint buffer when editing the window.

PAINT BUFFER AREA

This area is not compressed and can be broken into six byte structures.

PAINT STRUCTURE:

```
struct {
  char trow;                    /* top row of paint                */
  char tcol;                    /* top column of paint             */
  char erow;                    /* ending row of paint             */
  char ecol;                    /* ending column of paint          */
  unsigned colatr;              /* paint color attribute           */
  };
```

There will be at least one paint structure for each window. This paint structure indicates the base foreground and background colors for this window. For each different paint used in Helper, a corresponding paint structure will be in the paint buffer.

HELP FILE COMPRESSION

For each help window, the window buffer and the control buffer areas are compressed. We will explain the compression technique through illustration. First, we will decompress part of the window buffer for the HELP1 window in the help file dump. This should give a clear explanation of how the data is compressed.

11 byte compressed section of the HELP1 window buffer:

byte offset	1	2	3	4	5	6	7	8	9	10	11
hex value	01	C9	00	CD	3C	BB	BA	00	20	3C	BA
		╔		═		╗	║		SP		║

Ascii Characters to be Displayed

Byte	Hex value	Explanation
1	01	This indicates that the window buffer is in fact compressed. A value of (00) would indicate that the entire buffer is not compressed.
2	C9	This is the first character in the help window. Since it is not preceded by a (00), no compression took place on this character. Store (C9) '╔' to the uncompressed buffer.
3	00	Compressed character indicator. This tells us that the next byte is the value of a compressed character.
4	CD	This is the character value that was compressed.
5	3C	This is the number of positions that byte 4 occupied in succession. Store 60 (3C hex) '═' characters into the uncompressed buffer.
6	BB	This indicates no compression. Store (BB) '╗' to the uncompressed buffer.
7	BA	This indicates no compression. Store (BA) '║' to the uncompressed buffer.
8	00	Compressed character indicator. This tells us that the next byte is the value of a compressed character.
9	20	This is the character value that was compressed.
10	3C	This is the number of positions that byte 9 occupied in succession. Store 60 ' ' characters into the uncompressed buffer.
11	BA	This indicates no compression. Store (BA) '║' to the uncompressed buffer.

If we displayed the characters that we uncompressed, it would look like the following:

It's easy to see how much space is saved by this compression technique. 124 window characters were compressed down to 10 bytes.

When you understand how the data is decompressed, it should be easy to understand the compression technique. There are a couple of things to remember when compressing/decompressing Clarion help windows:

1. Characters are normally compressed when three or more are found in succession.

2. The maximum number of characters compressed in succession is 255. (255 is the largest value a byte can hold.)

3. The characters and attributes are compressed separately, yet in the same buffer, to increase the amount of compression. When the entire windbuf has been decompressed, the characters and attributes are then reunited, char, attr, char, attr, etc.

HELP WINDOW MENU ITEM

A menu item in a help window gives that window the capability to jump to a particular help screen. A help window can contain multiple menu items, although they cannot have both menu items and a chain. When a menu item is created for a window, the "chain to" option is ignored and the window chain structure is replaced with the following window menu structure.

WINDOW MENU STRUCTURE:

```
struct          {
  char mrow;                    /* beg row of menu item                    */
  char mcol;                    /* beg col of menu item                    */
  char mlen;                    /* length of menu item                     */
  char sel;                     /* select attribute */
  char ctl;                     /* character attribute                     */
  char flet;                    /* first letter of menu item               */
  char mwind[8]                 /* window name to jump to                  */
  };
```

A menu structure is detected by the chain variable in the window header structure. When this variable contains a value greater than (01) it indicates menu structures to follow. As stated earlier, the number of menu structures is computed as follows:

number of menu items = chain - 1

This is done because Helper interprets a chain value of 1 as a CHAINED window, rather than a window with MENU ITEMS. Therefore, a 2 in the chain byte means that there is 1 menu item on the window.