hypercard.org contact

HyperCard file format

Caveat Emptor!

Although originally intended by Bill Atkinson, the HyperCard file format has never been officially published. The instructions in this file are simply what was deduced by looking at various existing stacks and their differences.

Warning: The information in this document is not complete enough to allow the creation of new HyperCard stacks, but maybe it can be helpful in reading existing stacks and extracting precious data to keep it from being lost.

Prerequisites

Being a file format from Classic MacOS (shipped 1987 through 2004), many of the data types are from that era. All text is encoded in the MacRoman text encoding, and many flags and data types are from the Quickdraw headers, or based on them. All data is stored in Big-Endian format (like the Motorola 68000 used to do).

The block file layout

A HyperCard stack is a stream of blocks, with a four-character type code and a 4-byte signed ID number, terminated by a `'TAIL'` block. Each block has the following basic layout:

4 byte	Block size including size, type and ID.
4 bytes	Block type
4 bytes	Block ID number
n bytes	Block data

The Stack Block

There is one block in the file representing the stack object:

4 byte	Block size including size, type and ID.
4 bytes	Block type `'STAK'`
4 bytes	Block ID number `-1`
32 bytes	*There be Tygers*
4 bytes	Number of cards in this stack.

4 bytes	The ID of one card in the stack (why?).
4 bytes	The ID of the 'LIST' block in the stack.
16 bytes	*There be Tygers*
2 bytes	The User Level setting (1 ... 5) to run this stack under.
2 bytes	*There be Tygers*
2 bytes	Flags: Bit 10 is cantPeek, 11 is cantAbort, 13 is privateAccess, 14 is cantDelete, 15 is cantModify.
18 bytes	*There be Tygers*
16 bytes	Three 4-byte `NumVersion` entries containing the HyperCard version numbers that created, last edited or compacted this stack.
328 bytes	*There be Tygers*
2 bytes	The height in pixels of cards in this stack.
2 bytes	The width in pixels of cards in this stack.
262 bytes	*There be Tygers*
For each Pattern: (40 patterns, 320 bytes)	
8 bytes	Raw data for an 8x8 bitmap, with one byte representing one row.
512 bytes	*There be Tygers*
n bytes	Stack script as a C string, terminated by a NULL byte.

The Style Table

Styles for multi-styled text fields are stored as style formats in the style table block of the stack, and only referenced throughout the stack. Here's how the style table looks:

4 byte	Block size including size, type and ID.
4 bytes	Block type `'STBL'`
4 bytes	Block ID number
4 bytes	*There be Tygers*
4 bytes	Number of styles
For each style:	
16 bytes	*There be Tygers*

2 bytes	Font ID -1 to indicate the font should be inherited from the containing field's styles.
2 bytes	Style flags The high byte contains the standard Quickdraw font bits: bold, italic, underline, outline, shadow, condense, extend, group. If both bytes are `0xFF`, the style is unset and should be inherited from the field's styles.
2 bytes	Font Size -1 to indicate the size should be inherited from the containing field's styles.
2 bytes	*There be Tygers*

The Font Table

Since font IDs are not persistent across systems, HyperCard contains a table mapping font names to IDs, so it can get away with storing font IDs in the stack file, but still map the ID to the new one when it changes. The table looks like this

4 byte	Block size including size, type and ID.
4 bytes	Block type `'FTBL'`
4 bytes	Block ID number
6 bytes	*There be Tygers*
2 bytes	Number of fonts
4 bytes	*There be Tygers*

For each font:

2 bytes	Font ID
1 byte	Length of name
n bytes	Name of the font
0...1 bytes	Alignment byte

The Page Table List

Cards may be stored in an arbitrary order in a stack, so there is list of pages that, among other things, contains an ordered list of the card IDs in this stack. To speed up insertions/deletions of cards in large stacks, this list has been segmented. There is one 'LIST' block listing all page tables, and then a bunch of page table blocks listing the cards in order. Here's the list block's format:

4 byte	Block size including size, type and ID.

4 bytes	Block type `'LIST'`
4 bytes	Block ID number
4 bytes	Number of page tables
8 bytes	*There be Tygers*
2 bytes	Size of card blocks
16 bytes	*There be Tygers*
For each page table:	

2 bytes	*There be Tygers*
4 bytes	ID of 'PAGE' block

The Page Table

The page table stores the order of the cards in the stack (because the actual card data is kept in an arbitrary order in the file) and is segmented into several blocks. Each block looks like the following:

4 bytes	Block size including size, type and ID.
4 bytes	Block type `'PAGE'`
4 bytes	Block ID number
12 bytes	*There be Tygers*
For each card (get the count from the STAK block):	

4 bytes	Card ID
1 byte	Card flags, where bit 5 contains the marked of the card.
n bytes	*There be Tygers* (where *n* is the "Size of card blocks" from the 'PAGE' block, minus the 5 bytes for card ID and flags)

Cards and Backgrounds

Both cards and backgrounds are stored in the following kind of block:

4 bytes	Block size including size, type and ID.

4 bytes	Block type `'CARD'` or `'BKGD'`
4 bytes	Block ID number
4 bytes	*There be Tygers*
4 bytes	ID of BMAP block for card picture (0 means all transparent)
2 bytes	Flags. Bit 14 is cantDelete, 13 is hide card picture, 11 is dontSearch,
14 bytes	*There be Tygers*
4 bytes	ID of background for this card
2 bytes	Number of parts on this card
6 bytes	*There be Tygers*
2 bytes	Number of part contents on this card
4 bytes	Script Type: If 0 HyperTalk, if `'WOSA'` the card has a compiled OSA language script (e.g. AppleScript).

For each part:

2 bytes	Length in bytes of this part entry
1 byte	Type: 1: button 2: field
1 byte	Flags: Bit 7 is hidden, bit 5 is dontWrap, 4 is dontSearch, 3 is sharedText, 2 is fixedLineHeight, 1 is autoTab, 0 is disabled/lockText
2 bytes	Top coordinate of part rectangle.
2 bytes	Left coordinate of part rectangle.
2 bytes	Bottom coordinate of part rectangle.
2 bytes	Right coordinate of part rectangle.
2 bytes	More flags: bit 15 is showName/autoSelect, bit 14 is highlight/showLines, bit 13 is wideMargins/autoHighlight, bit 12 is sharedHighlight/multipleLines, bits 11 through 8 is the button family number Low 4 bits = style: Buttons: 0 is transparent, 1 is opaque, 2 is rectangle, 3 is roundrect, 4 is shadow, 5 is checkbox, 6 is radiobutton, 8 is standard, 9 is default, 10 is oval, 11 is popup. Fields: 0 is transparent, 1 is opaque, 2 is rectangle, 4 is shadow, 7 is scrolling.

2 bytes	titleWidth/lastSelectedLine
2 bytes	icon ID/(first)SelectedLine
2 bytes	textAlignment: 0 left (or default?), 1 center, -1 right, (-2 force left align?)
2 bytes	textFont ID
2 bytes	text font size
2 bytes	line height
2 bytes	text style flags: bit 15 is group, 14 is extend, 13 is condense, 12 is shadow, 11 is outline, 10 is underline, 9 is italic, 8 is bold
2 bytes	line height
n bytes	Name of the part as a zero-terminated C string.
n bytes	Script of the part as a zero-terminated C string.
0 ... 1 bytes	Aignment byte, if needed.

For each part content entry:

2 bytes	Part ID (of part these contents belong to) If this is < 0, this is an entry for a card part (take partID * -1), otherwise for a background part.
2 bytes	Length of this contents entry.
2 bytes	Length of styles If this is > 32767, there is stylesLength -32770 bytes of style data prepended to the text, otherwise the text is mono-styled.

Style data is *stylesLength / 4* entries like:

2 bytes	start offset
2 bytes	index into 'STBL' block

n bytes	Text data

n bytes	Name of the card as a zero-terminated C string.
n bytes	Script of the card as a zero-terminated C string.

Optional OSA script data:

2 bytes	Offset from end of this field to OSA script (jump across header)
2 bytes	Length of OSA script
n bytes	Remainder of header
n bytes	OSA Script

If you have found out more about this file format, feel free to amend this document and let us know.

Written up by Mister Z.

A day of acquaintance, And then the longer span of custom. But first -- The hour of astonishment.