Apple® IIGS

Spectrum™ 2.5.3 Reference Manual

Notes on Freeware Edition

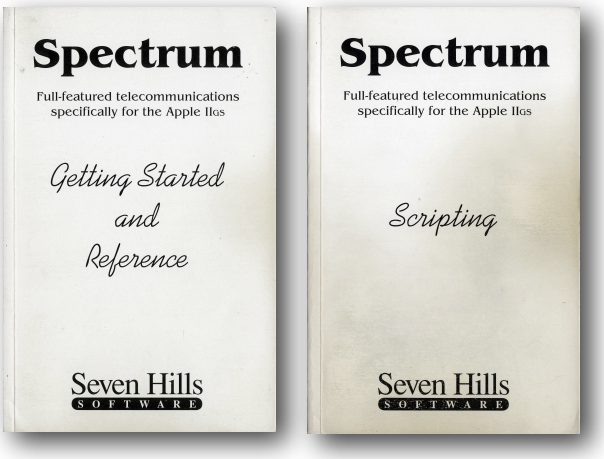
From an initial inception as a IIgs communications program called Impala, Spectrum was developed and released under the guidance of Dave Hecker of Seven Hills International. Originally released back in 1991, it was sold by Seven Hills until their eventual closure.

Seven Hills agreed I could continue to sell Spectrum, so I asked Eric Shepherd of Syndicomm to sell Spectrum on my behalf. In 2011 Tony Diaz took over Syndicomm and continued to sell it for me. With the declining sales over the years, I finally decided that the time had come in 2012 to make Spectrum Freeware.

To coincide with its reclassification, this PDF manual has been compiled from the original Seven Hills manuals. I have changed a few words and links here and there, and added new screen shots and illustrations.

Ewen Wannop - 2012

Spectrum



About Seven Hills Software

No Copy Protection

We don't believe in copy protection—all it does is impair the honest user's ability to use software to its fullest. We strive to provide high quality products at reasonable prices. We hope you will support our efforts by not allowing your family or friends to copy this software.

Questions and Comments

We always welcome feedback—if you have any questions, or suggestions for improving this product, please let us know. In addition, we would like to hear your ideas for new programs.

Copyrights and Trademarks

Table Of Contents

Spectrum Scripting

XCMDs

Programming Information

Working With Spectrum

From its inception, Spectrum was designed in a modular fashion, allowing Add.Ons such as port drivers, online displays, and from v2.0 onwards, XCMDs, and even XDisplays! These expanded on the powerful capabilities of Spectrum itself, allowing it to keep up with ongoing events such as Marinetti and the Internet.

Programming information for these Add.Ons was until now only available on request. With the release of Spectrum as Freeware, I am now making this information available to all.

XCMDs

Spectrum's scripting language is extremely powerful, and covered most all of the things we thought at the time people might wish to do. As time went on, we realised we needed more, so the XCMD came into being. The XCMD added additional capability to an already powerful application. An XCMD allows a script to pass control to an external module, which running at full code speed, can do most anything you can think of!

With the Browser XDisplay, we even created an online HTML display that thought it was an XCMD. This allowed rendering of HTML data passed to it from the Spectrum Internet Suite (SIS) scripts. The base code from the Browser XDisplay has now had a further lease of life within the HTML Tool Set.

You will find here the scripting documentation for the various XCMDs, including some of the legacy XCMDs, such as Freezer and TopCat. If you wish to write your own XCMD, check out the XCMD Programming Information, and the XCMD Sample Code at the end.

References & Parameters

If you want to know how to interface with Spectrum, either to write your own Online Display, or to interface from your own XCMD, check out the documentation at the end of this PDF.

For completeness, I have also included programming information for the !Help! NDA.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop January 1995
Updated by: Ewen Wannop May 1996

This technical note describes the BatchXfer XCMD

The BatchXfer XCMD

The BatchXfer XCMD adds batch Ymodem and Zmodem file transfers to script commands. This allows batch uploading of files to hosts which do not recognize automatic transfers. The BatchXfer can be controlled either directly from a list of files provided by a script, or by retrieving a list of files manually from an _SFMultiGet2 dialog.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

EXT BatchXfer 0 <VarName>

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Sends single or batch files)

Function 1 sends single or batch files:

EXT BatchXfer 1 <Method> {Prefix} “FileName1” {“FileName2”} etc.
EXT BatchXfer 1 <Method> {Prefix} {EditorHandle}
EXT BatchXfer 1 2 10 “File1” {“File2”} etc.
EXT BatchXfer 1 2 “:Ram5:” “File1” {“File2”} etc.
EXT BatchXfer 1 2 ~E01234

If the optional Prefix is present it will be added to each file to make the complete path for sending the files. Files are first checked for the presence of a full path indicated by a colon within the FileName. The Prefix is not added if a full path is found. The Prefix can be a number, or a path enclosed by colons. If the Prefix is a number, the script should first set this prefix by using ‘Set GSPrefix <PrefixNumber> “FolderName”’ before making the XCMD call. Any prefix may be used that is not reserved. Spectrum does not directly use any of the prefixes above 10.

If the optional ScriptEditorHandle is present, the command will use this as the source of FileNames. The Handle is checked to see if it is valid, the Failed flag being set if it does not exist. ScriptEditorHandles are in the form of ‘~E01234’. Use the script substitution ‘$EditorHandle#’ to retrieve a ScriptEditorHandle as input to this command. If a ScriptEditorHandle is given, the rest of the command line will be ignored. If the optional Prefix is present, this will be added to the filenames extracted from the EditorHandle.

The input to the EditorHandle may be from any of the usual sources. FileNames must be bracketed by the current QuoteCharacter if they contain any spaces. If they are not bracketed, the FileNames will be delimited by the first space character or carriage return. Carriage returns and linefeeds may be freely used between entries.

If the optional EditorHandle is not given, the FileNames listed on the command line will be sent. If the optional Prefix is present, this will be added to the filenames extracted from the EditorHandle. As many files can be listed as will fit into the script command.

Note: The current Quote character must be used to enclose filenames.

The<Method> indicates:

1 Ymodem batch upload
2 Zmodem batch upload

Valid filenames must be specified. If a file does not exists, or has a resource fork, the transfer will be stopped and the Failed flag set, any remaining files will not be sent. You may determine how many files were sent sucessfully by calling Function 3.

Function = 2 (Send Multiple Files)

Function 2 allows manual selection of files:

EXT BatchXfer 2 <Method>

A standard _SFMultiGet2 dialog will be shown. Multiple files can be selected from this dialog.
If the SHR desktop is not showing when this call is made, the call will do nothing and the Failed flag will be set.

The<Method> indicates:

1 Ymodem batch upload
2 Zmodem batch upload

If Spectrum v2.1 or later is running, and Send forked files by MacBinary is turned on, forked files will display in the dialog. If an earlier version of Spectrum is running, or MacBinary is turned off, forked files will display dimmed.

Function = 3 (Get File Count)

Function 3 returns the number of files that were sent in the last Xfer:

EXT BatchXfer 3 <VarName1> {VarName2}

VarName1 will be set to the number of files that were successfully sent in the last BatchXfer.
The optional VarName2 will be set to the result of the last BatchXfer:

If no errors occurred VarName2 will be set to a null string
If an error occurred in the BatchXfer VarName2 will be set to ‘1’

Spectrum XCMD Technical Notes

Written by: Ewen Wannop
Updated by: Ewen Wannop

March 1996
September 2000

This technical note describes the BinHQX XCMD

The BinHQX XCMD

The BinHQX XCMD gives a script control over the encoding/decoding process associated with BinSCII (Apple II), BinHex 4.0 (Macintosh), MIME (Base64) and UUencoded files. For versions of Spectrum later than v2.1, full manual control is also provided through menu items inserted into the Spectrum menus.

All four encoding formats allow the transfer of 8 bit native data files over a 7 bit ASCII network. The 8 bit data is reduced to printable ASCII text and the transmission can therefore ignore any control characters introduced within the transmitted data other than CR's or LF's. This makes the encoded data robust enough to be stored and transmitted on all current networks and host systems.

Typically BinSCII, BinHex 4.0, MIME (Base64) and UUencoding are used to send files as attachments to messages over Internet message systems.

BinSCII is specifically tailored to the Apple II GS/OS (ProDOS) file system and preserves the filetype and auxtype information of these files. Because it was defined before extended files were created for use with the IIgs, it does not allow the encoding of forked files. Such files must first be packed into a single data fork using an archiving program like ShrinkIt.

BinHex 4.0 is specifically tailored to the Macintosh file system and preserves the Type and Creator of the Macintosh file, as well as both the data and resource forks of the file. However BinHex 4.0 can also be used to pack forked GS/OS files as well as Macintosh files. This allows forked files to be directly encoded unser the GS/OS file system.

The MIME (Base64) format is becoming more common than UUencoding, and has the advantage that it can fully support AppleSingle and AppleDouble format. This allows both forks of a file to be preserved with its filetype attributes. Optionally, Base64 encoding can be applied to the Data fork only. In this case, no file attributes will be preserved. This method is more suited to plain text files, or files, such as ShrinkIt archive files, that do not care what filetype they are. The correct MIME headers required for email systems are also added with this encoding. The separate sections of any split files may need to be joined together before decoding, depending on the capability of the decoding application.

UUencode can only handle data forks, and does not include any file information. It is commonly used for systems that do not require filetypes, where the filename suffix is normally used to distinguish the file type. It may be encountered in files enclosed in email and Usenet News Groups.

Function = Null (Set Parameter dialog)

With no function specified, and with the SHR 640 desktop showing, a dialog window will open allowing you to manually set the adjustable settings that are used for Function 3 and the Menu control of the encoding process.

EXT BinHQX

The 'Save' button will save the settings to the file 'BinHQX.Prefs' in the Add.Ons:XCMDs folder. If this file exists when BinHQX is next loaded, these saved values are used as the new defaults. You can restore the factory default parameters with the 'Defaults' button. The 'Cancel' button will ignore any changes to the settings since the dialog was opened.

Note: These settings are not used for the script control in Function 1. The settings must be given there within the supplied parameters as described in the Function.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

EXT BinHQX 0 <VarName>

The VarName, if present, will return the version number of the XCMD.

Function = 1 (EncodeFile)

Function 1 encodes a file:

EXT BinHQX 1 Kind Parts EndLine "SourcePathFileName" "OutputPath" {OutputName}
EXT BinHQX 1 Kind Parts EndLine EditorNumber EditorNumber {OutputName}
EXT BinHQX 1 Kind Parts EndLine ~E01234 ~E01234 {OutputName}
EXT BinHQX 1 Kind "IdentifyString" EndLine ~E01234 ~E01234 {OutputName} *

Where Kind:
1 = Encode BinSCII
2 = Encode BinHex 4
3 = Encode MIME (Base64) AppleSingle or AppleDouble
4 = Encode UUencode
5 = Encode MIME (Base64) Plain data files
6 = Encode MIME (Base64) AppleSingle or AppleDouble *
7 = Encode MIME (Base64) Plain data files *

* These options write minimum headers for use with smart mailers.

Where Parts:
0 = Single output file
N = Number of parts in output file

Where EndLine:
10 = LineFeed ($0A)
13 = Return ($0D)
09 = Tab ($09) (BinHex 4.0 only)
32 = Space ($20) (BinHex 4.0 only)

In the case of a BinSCII output file, the body of the file is split into 12K segments. If 'Parts' is any number other than '0', the output file(s) will consist of that number of segments (or less if the file is shorter). A number of '1' will result in one file for each segment.
For BinHex 4 and Base64 output files, if 'Parts' is any number greater than '24', then the file will be

segmented. In this case ‘Parts’ denotes the number of lines per segment. Each segment is always saved as a single file.

EndLine denotes the method used to separate each line in the output file. Most systems use a standard CR ($0D) as their end of line character. The Macintosh and the Apple II both use this value. Some systems, such as Unix, expect an LF ($0A) as their end of line character. EndLine is given as the ASCII value of the required character. The only allowable values for BinSCII and Base64 are 10, and 13 and those for BinHex 4.0 are 9, 10, 13 and 32.

The input Pathnames to this command can be a “:Path:FileName”, EditorNumber or EditorHandle. Only a single file can be encoded in one pass. The output may be a single file, or a series of files, depending on what values for Kind and Parts were chosen. The output path may be given as a “:Path”, “:Path:FileName”, EditorNumber or EditorHandle.

When encoding a file, the OutputPath will be used to store the encoded file(s) using a name based on the input filename. The OutputPath may be given as a full path or a prefix. To signify the end of the path, a ‘:’ is obligatory. If this is not present an error will be created.

If the optional {OutputName} variable is given, it will return the first or root name used for the output file.

Note that for Kind options 6 & 7, the file is always written as a single output file, and instead of supplying a number for Parts, a 24 character “IdentifyString” must be supplied. This should be unique for each message and will be used to create the boundaries between message parts.

Note that BinHex 4.0 was originally intended for encoding Macintosh forked files. As a Macintosh file can have an empty data or an empty resource fork, ProDOS or GS/OS data files encoded with BinHex 4.0 may have an empty resource fork when they are decoded. This may or may not be a problem if used within the GS/OS environment. It is not a problem for files that are decoded on a Macintosh. If you are encoding ProDOS or GS/OS files with only a data fork, it is usually better to use the BinSCII method.

The BinHQX XCMD will attempt to save ProDOS files with only a data fork, without a corresponding resource fork. Whether this is successful or not depends on how the file was encoded into BinHex 4.0 format in the first place. If encoded with the BinHQX XCMD, or the ProDOS file was encoded by a Macintosh application from an HFS partition, then the files should decode without a resource fork.

If an error occurs in the input parameters, or a disk error occurs, the Failed flag will be set.

Function = 2 (DecodeFile)

Function 2 decodes a file or list of files:

```
EXT BinHQX 2 “SourceFile” “OutputPath” {OutputName}
EXT BinHQX 2 EditorNumber EditorNumber {OutputName}
EXT BinHQX 2 ~E01234 ~E01234 {OutputName}
```

The source file may be a BinSCII, BinHex 4.0, MIME (Base64) or UUencoded file. If a series of file names is to be given, the names must be passed in a ScriptEditor handle. They will then be decoded in the sequence they are given. This is not a problem for BinSCII, but could cause problems with BinHex 4.0 or Base64 if the data in the files was not sequential.

The input to this command can be a “:Path:FileName”, EditorNumber or EditorHandle. The output path may be given as a “:Path”, EditorNumber or EditorHandle. The decoded file will be saved to the OutputPath using the embedded filename. If the optional {OutputName} variable is given, it will return

the name used for the output file. This will be the last file, if more than one output file is found within the source files.

Note that BinSCII encoded files can contain one or more segments of the final output file. The decoding process will decode each segment as it is found and save this to the target file. If not all the segments are present, the resulting file may be incomplete. If the target file name already exists, but is not the real target file, then any data it contains may be overwritten. There is no way of checking for this as the normal operation of BinSCII decoding is to write portions of the file to the targert file as they are found. It is wise to make sure you decode to an empty directory, or a directory that only contains known files.

BinHex 4.0 split files are handled differently. A split is designated by the following sequence. This can occur one or more times within a single file, or may be used at the end and start of two or more files. It is important that the files are seen and processed sequentially, or the data will be seen as being corrupted.

previous hex data...
<return>--- end of part xx ---<return> (processing stops)
any data found here is ignored...
<return>---<return> (processing continues)
new hex data...

The <return> in this case is any occurence of a <CR> or <LF>. Data between the two markers is ignored.

MIME Base64 files are split with appropriate headers added, so the files may be transmitted and reassembled later in the correct sequence. It may be necessary to actually join the pieces together before decoding depending on the capabilities of the decoding application.

UUencoded files are split with a 'Cut Here' header indicating which segment each file represents. It may be necessary to join the split files before decoding.

If the SourceFile is neither a valid BinSCII, BinHex 4.0, Base64 or UUencoded file, or a disk or other error occurs, the Failed flag will be set.

Function = 3 (ManualEncode)

Function 3 displays Standard File dialogs to choose the source file and target folder:

EXT BinHQX 3 {OutputName}

Choose the encoding method from the radio buttons in the dialog window. The settings used for Split files and segment sizes are obtained from the saved settings or settings dialog. This dialog can be called up from the Null Function, or from the Menu item. The 'Base64 Apple' checkbox, controls whether an AppleSingle or AppleDouble file is encoded, rather than a plain Base64 file from the Data fork only. (See Function 1 for further details on the encoding process.)

If the optional {OutputName} variable is given, it will return the name used for the output file.

If disk or other error occurs, the error will be displayed in an Alert window, and the Failed flag will be set.

Function = 4 (ManualDecode)

Function 4 displays Standard File dialogs to choose the source file(s) and target folder:

EXT BinHQX 4 {OutputName}

The source file(s) are selected from the dialogand will be decoded in the sequence they are found within the dialog. (See Function 2 for further details.)

If the optional {OutputName} variable is given, it will return the name used for the output file.

Only TXT, SRC and TEA files will normally show in the dialog, but for compatibility with Macintosh sourced files, those with a filetype of $00 will also show.

If the SourceFile is neither a valid BinSCII, BinHex 4.0, Base64 or UUencoded file, or a disk or other error occurs, the error will be displayed in an Alert window, and the Failed flag will be set.

Menu Selection

From Spectrum v2.1 onwards, XCMDs have been able to insert menu items directly into the Spectrum menus. The BinHQX XCMD will insert two items into the (Extras) menu, and one item into the Transfer Settings menu.

'Encode File...'	is similiar in action to Function 3
'Decode File...'	is similar to Function 4
'Encode Settings'	is the same dialog that the Null Function shows

This allows the BinHQX XCMD to be used in three different ways. The XCMD can be fully controlled by conventional EXT script commands, or can be used by calling up the manual and file dialogs from the script commands. Optionally, for versions of Spectrum from v2.1 onwards, it can be completely controlled from the inserted menu items without requiring any script action.

Any errors generated during the encoding or decoding process are displayed in a standard Alert window.

Balloon

After a file has been decoded, an IPC messages is broadcast by the BinHQX XCMD in the same way that Spectrum notifies a successfully downloaded file. If Balloon, or a similar program is present, it will respond as if the file had been downloaded. In the case of multiple BINSCII files being decoded in one session, the IPC message will only refer to the last file to be decoded.

Note that Balloon will not process the list of files it has been passed until Spectrum Quits.

Scripting Examples

In order to decode multiple files using a script command, it is necessary to pass a list of files within a ScriptEditor handle. When the handle is parsed by the BinHQX XCMD, it considers each file entry as being separated by the current Quote character or by spaces. If you are using the Quote character, you need to construct the ScriptEditor with strings, using the current Quote character. Do this by simply changing the Quote round the construction process:

```
Set Quote •
Create ScriptEditor 0
Ext ScriptEditor 7 0 •":Path:File1" ":Path:File2" ":Path:File3"•
Set Quote "
Ext BinHQX 2 0 "OutputPath" {OutputName}
```

Spectrum XCMD Technical Notes

Written by: Ewen Wannop
Amended by: Ewen Wannnop

January 16th 1999
July 22nd 2003

This technical note describes the Browser XDisplay

The Browser XDisplay

The Browser XDisplay is a hybrid online display and XCMD. It has a file type of an online display, and therefore should be placed in the Spectrum:Add.Ons:Online.Displays: folder. It is loaded at Spectrum boot as a normal display. If then opened as an online display, it will display a window with a TextEdit control, and optional tool bar, filling the screen. It will then respond to IPC calls under the XCMD name of 'Browser'. It can only fully be used in conjunction with a suitable control script.

The Browser XDisplay may be opened and closed as you would normally expect with an online window, and all data sent to the display is returned unchanged to the capture buffers. Only if the data includes appropriate HTML structures will it be displayed on the screen. There is also a separate function to display plain or HTML text files from Scripteditor Handles. Key presses, except for Escape and OA-R, are routed normally to the port, though this function may be turned off. The Browser XDisplay will show and respond as ‘Web Browser’ on the Spectrum’s Online Display settings list. Once it has been opened, the Browser will respond to normal XCMD calls. When closed, it will not respond to these calls.

When Function 1 has been called, the display will start to capture data. When a complete page has been received, a TextEditor display is built with either a simple HTML display, or just Plain text. The Browser XDisplay can then be interrogated with various XCMD calls. If the page length extracted from the header is passed to Function 1, then a progress thermometer will be shown as the page is loaded, otherwise a barberpole will be shown.

Note that if HTML is sent through the Display script command, it will bypass the check for '<body>'. This allows HTML to be sent to the display and be immediately displayed by calling Function 14.

The Browser display uses part of the MenuBar to display progress messages (See Function 6). Typically this will leave room for five or six menu names as well as the Apple Menu. The messages start at a position of 380 pixels and run through to the Spectrum clock display. Scripts should set the StatLine OFF to leave space for these messages. At the time of writing, the ID of the TextEdit Control was $7001. This value can be passed to the WindowMgr if required.

Versions of the Browser from v1.4 onwards support inbound 'Inter XCMD Communication'. This allows faster responses where one XCMD wishes to call another through direct IPC calls, rather than through a Spectrum controlled script. See the main XCMD documentation for more details.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

EXT Browser 0 <VarName>

The VarName, if present, will return the version number of the XCMD.

Note that you will not normally be able to display the resulting VarName on screen unless you close the 'Web Browser' display and then open a normal display.

Function = 1 (NewPage)

Function 1 clears memory of the current page allowing the module to scan for a new page in the input data stream:

EXT Browser 1 <value1> {value2}
EXT Browser 1 <MIME-type> {length}

Where:

MIME-type = 1 for 'HTML' and 2 for 'plain text'
{length} length value extracted from page header

The currently displayed TextEdit Handle will not be changed until a new page has been fully received. This Function must be called before each new page that you wish to receive. The optional {length} value should be extracted from the header. If it is not provided, or has the value of '0', then the barber pole will be shown instead of the thermometer.

Call Function 14 to complete the capture and display the page. If the 'Connection closed' or '|Telnet| Returning' string is present at the end of the capture, it will be removed.

The 'null event' counter is reset by this call. For approximately every 30 seconds that no events occur, any Suspended script will be woken up and the Kind variable set to 0. The actual timing depends on the clock speed of your IIgs.

The Failed flag will be set if any error occurs.

Function = 2 (SetHit)

Function 2 sets the response to a hit on the TextEdit control or Tool bar:

EXT Browser 2 Action <VarName1> <VarName2> {VarName3} {VarName4}
EXT Browser 2 1 Kind String {Name} {Base}

Where:Action = 0 to turn off screen response (variables not required)
Action = 1 to turn on

Returned values:

Kind = 0 No activity for about 30 seconds, variables unchanged
Kind = 1 String holds URL of a page
Kind = 2 String holds URL of an image
Kind = 3 Page has been completed, String holds META CONTENT string *
Kind = 4 Page header has been received, page arriving, variables unchanged
Kind = 5 GET Submit button pressed, String holds Form number **
Kind = 6 POST Submit button pressed, String holds Form number **
Kind = 7 ToolBar button hit, String holds Button number
Kind = 8 Linked, String holds URL of link.
Kind = 9 Ctrl-Double-Click on a non-linked part of the page
Kind = 10 Ctrl-Double-Click, String holds URL of a page
Kind = 11 Ctrl-Double-Click, String holds URL of an image
Kind = 12 Ctrl-Double-Click, Linked, String holds URL of link.

The ToolBar Button numbers are:

Back = 1
Home = 2
Forward = 3
Reload = 4
Save = 5
Open = 6
Print = 7
Find = 8
Stop = 9

These button hits are only reported, it is up to the script to activate any related action.

* If the page holds a META HTTP-EQUIV="refresh" tag the String variable will be filled with the CONTENT field. Otherwise String will be empty and unchanged. The script will need to parse the resulting String and set a script Timeout. The String value is returned exactly as it was found in the META tag. This will usually have a number value to start, with an optional URL following.

Depending on the CPU speed of your IIgs, if there has been no activity for about 30 seconds after calling Function 1, the system null events will trigger a resumption of the script. Kind will be 0.

The Kind variable is set to 1 if a double click is made on a an underlined reference in the displayed page, and the String variable will return this reference as a string. It is set to 2 if an Image icon is hit and the reference for the image is returned.

If the optional Name variable is given, and if Kind is 1 or 2, and a Name link was referenced, then it will hold the Name of that reference. It will be empty if there is no Name link. A script should then call Function 10 with this variable when the new page has been retrieved and drawn.

** If the optional Base variable is given, in addition to the Name variable, and the current document has a <BASE HREF=...> embedded attribute, and if Kind is 1 or 2, this variable will be set to that Base address. It will not be changed if there is no Base reference. It can be used to build complete URLs from partial ones.

The Submit buttons will return a number value in String. Use this number as input to Function 15.

The SetHit function should be used in conjunction with the 'Suspend Script' script command for the most flexible response. The variables will then be set and reported in the script stream. VarName1 will be set to the Kind of hit that was last made on the TextEdit control, String to the associated string for that hit, Name to any forward referenced link and Base to any Base address.

The Failed flag will be set if any of the required parameters are invalid or are not present.

Function = 3 (GetSource)

Function 3 returns the raw page data, or data that was last loaded from a file, or optionally the processed display, into the supplied TEHandle:

EXT Browser 3 <TEHandle> {Value}
EXT Browser 3 ~E01234 {1}

Without the optional Value, the raw page text of the currently displayed page will be returned. The optional value should set to '0' for the raw page text and to '1' for the processed display. The ScriptEditor Handle may then be saved or processed normally.

The Failed flag will be set if any of the required parameters are invalid or are not present, or raw text was requested and the display had been loaded directly from a Teach file.

Function = 4 (SetSource)

Function 4 builds a new page from the supplied source data in TEHandle:

EXT Browser 4 <TEHandle> {Value} {String}
EXT Browser 4 ~E01234 {1} {"Name"}

The source data should be loaded from disk, and then passed in the TEHandle. The data will then be displayed in the TEHandle supplied from Function 1, as if the page had come normally from the port. The optional value should be set to '0' to set raw source data in this way, or to '1' if the ScriptEditor holds Teach or processed data which should go directly to the display.

If the optional Name is present the Browser will attempt to jump to that Name in the body text. See Function 10 for more details.

The Failed flag will be set if any of the required parameters are invalid or are not present.

Function = 5 (GetTitle)

Function 5 gets the Title of the current page:

EXT Browser 5 <VarName>
EXT Browser 5 Title

The page Title extracted from the HTML data is not normally displayed within the page. This function will retrieve the Title for use with Bookmark functions etc. If no page is currently being displayed, or there was no <Title> in the data, the Failed flag will be set and the <VarName> will be set to empty.

The Failed flag will be set if any of the required parameters are invalid.

Function = 6 (PostMessage)

Function 6 posts a message in the menu bar from Pixel 380:

EXT Browser 6 <Value> {Value} or {String}
EXT Browser 6 1 12
EXT Browser 6 2 "[Host not responding]"

The Values 1 and 2 are special, the others will display a standard message:

```
Value     = 0  [                        ] a blanking message
          = 1  Displays a thermometer with {Value} ranging from 0-15
          = 2  Displays {String} which can be up to 22 characters long
          = 3  [Host contacted...    ]
          = 4  [Host unkown          ]
          = 5  [Getting page...      ]
          = 6  [Getting picture...   ]
          = 7  [Getting link...      ]
          = 8  [Page complete        ]
          = 9  [Drawing page...      ]
          = 10 [Function Failed...   ]
```

The Browser displays the thermometer during file loading or when receiving a page. If these messages are required, they must be displayed by the controlling script.

The Failed flag will be set if any of the required parameters are invalid or not present.

Function = 7 (UpdateScreen)

Function 7 updates the screen display from the internal raw data buffer:

EXT Browser 7

During certain script commands the Browser window may be closed and opened again round the call. If there is data in the text source buffer, this will be displayed once more on screen. If under a specific sequence, the Browser TextEdit control does not update, and is left showing an empty screen, this command will redraw the control once again from the data in the text source buffer.

Function = 8 (DisableToolButtons)

Function 8 enables and disables the Tool bar buttons:

EXT Browser 8 <Value>

Sets the state of the tool buttons bitwise from the supplied value. Set the bit to enable the button and clear it to disable. A value of 0 will disable all the buttons, and 511 will enable all the buttons. The value is given in decimal.

Bit 0 = Back
Bit 1 = Home
Bit 2 = Forward
Bit 3 = Reload
Bit 4 = Save
Bit 5 = Open
Bit 6 = Print
Bit 7 = Find
Bit 8 = Stop

The Stop and Open buttons share the same space in the tool bar. Priority is given to the Open button, and so this will always show if it is active, or it is inactive and the Stop button is also inactive.

The Failed flag will be set if any of the required parameters are invalid or not present.

Function = 9 (ShowSource)

Function 9 builds a new page from the supplied raw text source in TEHandle:

EXT Browser 9 <TEHandle> {Family} {Size} {Style} {Color}
EXT Browser 9 ~E01234

If any of the optional font values are not supplied, the default WWW-1 font will be used in 10pt black. Any of the optional values may be omitted, but all the previous values in the line must be supplied. See the Spectrum Script Addendum 'Draw Text' command for more details on the range of these parameters.

The raw text is displayed 'as is' and no attempt is made to interpret the contents. This is provided as a means of displaying pages that do not contain any HTML structures.

The Failed flag will be set if any of the required parameters are invalid or not present.

Function = 10 (GotoName)

Function 10 jumps to an internal name link:

```
EXT Browser 10 <String>
EXT Browser 10 "Name"
```

If an embedded name, in the form <A NAME="Name">, exists in the currently displayed document, then the screen will be scrolled to bring that point of the display to the middle of the screen. This is the same as clicking on a 'Goto' link icon.

The search is case sensitive, but does not respect the setting of the script CaseSensitive flag.

When a hit has been made on the screen, if the Name variable has been set, the variable will hold the reference to an internal link on the newly referenced page. The script should call this Function with that Name variable when the new page has been displayed.

Function = 11 (FindString)

Function 11 jumps to a target string:

```
EXT Browser 11 <Value> <String> <VarName>
EXT Browser 11 Start "TargetString" Result
```

The Value should be set to '0' to start searching at the top of the displayed page. Use the returned Result position as the next start value to Find the next occurrence of the target String. If the String is found, the screen will be scrolled to bring that point of the display to the middle of the screen, and the Result variable is set to the next character position after the string. If the String is not found, the Result variable will be set to '0'. The script can then act accordingly on this result.

The setting of the script CaseSensitive flag will be observed.

The Failed flag will be set if any of the required parameters are invalid or not present.

Function = 12 (SaveFile)

Function 12 saves the display to a Teach file on disk:

```
EXT Browser 12 "FolderNameFileName"
EXT Browser 12 ":Disk:Folder:File"
```

If there is a Title linked to the screen display it will be saved as resource type $C003 ID $0001, and if the raw page test is also present, it will be saved as resource type $C003 ID $0002. Active forms will be preserved.

The Failed flag will be set if any of the required parameters are invalid or not present, a GS/OS error occurs, or the display is empty.

Function = 13 (LoadFile)

Function 13 loads the display from a Teach file on disk:

```
EXT Browser 13 "FolderNameFileName" {String}
EXT Browser 13 ":Disk:Folder:File" {"Name"}
```

If resource type $C003 ID $0001 is present it will be loaded as the screen Title, and if resource type $C003 ID $0002 is present it will be loaded as raw page text. Active forms will be restored.

If the optional Name is present the Browser will attempt to jump to that Name in the body text. See Function 10 for more details.

The Failed flag will be set if any of the required parameters are invalid or not present or a GS/OS error occurs.

Function = 14 (ShowPage)

Function 14 stops port input and displays the received page:

```
EXT Browser 14 {String}
EXT Browser 14 {"Name"}
```

When using Telnet, the script should monitor for 'Connection closed by foreign host.', and then call this function. With connections other than Telnet, which do not return this string, call this Function when the page is complete. In the case of a stalled page, or you decide to time out, also call this Function. It is suggested in such a case that you send the following string to the display using the Display script command before you call Function 14:

```
Display "> <body><hr><h3>Transfer Interrupted!</h3>"
```

If the optional Name is present the Browser will attempt to jump to that Name in the body text. See Function 10 for more details.

If the cursor was frozen by Function 24, it will be unfrozen by this command.

The Failed flag will be set if any error occurs.

Function = 15 (GetFormData)

Function 15 returns the Form data from a displayed page in the supplied TEHandle:

```
EXT Browser 15 <FormNumber> <TEHandle1> <TEHandle2> <VarName>
EXT Browser 15 1 ~E01234 ~E04321 Type
```

The supplied TEHandle1 will be filled with the target URL. If the ACTION field was not present, or the ACTION field was empty, so no URL was supplied, the returned EditorHandle will be null. TEHandle2 will hold the return string required by both the GET and POST buttons of an active Form. VarName will hold either ‘GET’ or ‘POST’. The controlling script should then handle the interaction with the host accordingly.

If VarName holds GET, then send the return string after the URL separated by a ‘?’. If VarName holds POST then send the return string in the body of the submission.

If there is no Form with the number ’FormNumber’ active, or the screen was not drawn from HTML data, or any other error occurs, the Failed flag will be set.

Function = 16 (SendScriptEditor)

Function 16 sends a TEHandle directly to the port:

```
EXT Browser 16 <TEHandle> {BreakCount} {LineBreakChar}
EXT Browser 16 ~E01234 {255} {13}
```

The supplied TEHandle will be sent directly to the port in a constant stream without being processed in the usual way by Spectrum. The only Spectrum settings that will be relevant are the Xon/Xoff flag and the baud rate setting. This differs from the ‘Send SciptEditor’ command which is passed through various filters and other settings within Spectrum.

If the optional Breakcount is given, the flow will have the optional LineBreakChar inserted at the break count position. If the optional LineBreakChar is not given, then a return/linefeed pair will be sent at the break. Both values are given in decimal, with the BreakCount in the range 0-65535 and the LineBreakChar in the range of 0-255.

If the TEHandle is not a valid handle, or any error occurs, the Failed flag will be set.

Function = 17 (OpenCaptureFile)

Function 17 opens a capture file to receive incoming data:

```
EXT Browser 17 "PathNameFileName" {filetype} {auxtype}
```

Spectrum does not save all data into its capture buffers even with its flag settings optimally adjusted. Notably nulls are removed from the data stream. Much of the data processing is made after the data has been returned by the display, so calling this function will allow a more controlled capture of incoming data. It is intended to be used by scripts capturing embedded graphics within an HTML page, or for FTP utilities. The 'Trigger' and 'WaitFor' script commands are still active, so to use these functions, a script should run something like this:

```
Set Screen OFF
Ext Browser 17 "PathFilename"
WaitFor String "GraphicTriggerSequence"
Set Screen ON
WaitFor String "EndOfGraphicSequence"
Ext Browser 18
```

To ensure a clean and fast capture, some of the flag settings in Spectrum will be changed when Function 17 is called. They will be restored on calling Function 18, or when the script stops. The script should not change any of these flags itself between the two calls. The following settings are turned off, Auto-receive of B+ and Zmodem, Strip Line-Feeds and Echo. In addition, nothing is saved in the scrollback or capture buffers while capturing data. Although it will not directly interfere with data capture, it would be a good idea to also stop the AutoLearn function if it was active.

A Text file will be created unless the optional {filetype} is given. A value of zero will be used for the auxtype unless the optional {auxtype} is given.

The Failed flag will be set if any of the required parameters are invalid, not present or a GS/OS error occurs.

Function = 18 (CloseCaptureFile)

Function 18 close the open capture file:

```
EXT Browser 18
```

If no file has been opened, this command will do nothing, otherwise it will close the file and restore the altered flag settings within Spectrum. This call will be made automatically when a running script stops.

Function = 19 (SetBrowserPrefs)

Function 19 sets the Browser preferences:

```
EXT Browser 19 <Value>
```

A Value of 0 will reset to the default settings. The preferences are set bitwise by the values:

Bit 0 = 1	Tables are formatted
Bit 1 = 1	Tool bar is displayed
Bit 2 = 1	Report Page Done for Function 4 *
Bit 3 = 1	Disable keypresses **
Bit 4 = 1	Do not pass data ***
Bit 5 = 1	Call Function 14 if the Stop button is pressed****

* If the script supports 'META refresh', then you should set this bit if you want to know if there is an embedded META tag in HTML that you have just displayed by Function 4.

** Normally the Browser allows keypresses, other than Escape, OA-R and any others normally trapped by an inactive TextEdit, to be sent to the port. Setting Bit 3 stops key presses from being sent. Most implementations of a Web Browser would have this setting OFF.

*** If this bit is set, then data sent to the Browser for display will not be passed back to Spectrum. It will not then be stored in the Scrollback Buffer if 'Set ScrollData Filtered' has been set.

**** If the 'Stop' button is 'hit', then if Bit 5 has been set, Function 14 will be effectively called to Show the Page. The 'Hit' will still be reported if Function 2 is ON.

The Failed flag will be set if any error occurs.

Function = 20 (SetLineEditText)

Function 20 sets the content string of the ToolBar LineEdit:

```
EXT Browser 20 "String"
```

The Failed flag will be set if the ToolBar is not active, or any error occurs.

Function = 21 (GetLineEditText)

Function 21 gets the content string of the ToolBar LineEdit:

```
EXT Browser 21 <VarName>
```

The Failed flag will be set if the ToolBar is not active, or any error occurs.

Function = 22 (EncodeBase64)

Function 22 encodes the supplied ScriptEditor into Base64 format:

```
EXT Browser 22 <SourceTEHandle> <TargetTEHandle> {BreakChar(s)}
EXT Browser 22 ~E01234 ~E04321 {13}
```

The encoded text will be returned as one long block of Base64 characters. If the optional {BreakChar(s)} is given, then the encoded string will be broken after every 76 characters with the BreakChar(s).

The long decimal value for BreakChar(s) will be converted to Hexadecimal internally, so to give two BreakChars, such as CR/LF, you supply a value in excess of 256 with the low byte being inserted first. The decimal equivalent of CR/LF ($0A0D) is therefore 2573. You can use HodgePodge functions of you need to convert Hexadecimal numbers to decimal.

The Failed flag will be set if any error occurs.

Function = 23 (DecodeBase64)

Function 23 decodes the supplied ScriptEditor from Base64 format:

```
EXT Browser 23 <SourceTEHandle> <TargetTEHandle>
EXT Browser 23 ~E01234 ~E04321
```

The Failed flag will be set if any error occurs.

Function = 24 (FreezeCursor)

Function 24 disables the cursor:

```
EXT Browser 24
```

The Failed flag will be set if any error occurs.

Function = 25 (UnfreezeCursor)

Function 25 unfreezes the cursor:

```
EXT Browser 25
```

While the cursor is frozen, pressing the space bar will unfreeze it again. The cursor will always be unfrozen when the Browser window is closed.

The Failed flag will be set if any error occurs.

Function = 26 (ClearScreen)

Function 26 clears the screen and the raw data buffer:

```
EXT Browser 26
```

If a script stops while the display is still open and active, the display buffers and screen will be clear when the Browser next opens. If however, a script switches screens before it stops, this cannot happen. This function will allow the screen and buffers to be cleared at will. If the buffers are not cleared, calling Function 7 on what appears to be an empty screen, will restore the last screen that was displayed.

The Failed flag will be set if any error occurs.

Function = 27 (GetFileName)

Function 27 returns a file name from a Standard Get File dialog:

```
EXT Browser 27 "PromptString" <VarName>
EXT Browser 27 "Open which file?" FileName
```

If the script command 'Get File' is used, the Browser will close and reopen round the call. This Function avoids that from happening. Only TEXT, SRC and TEACH files will be shown.

Note that the Prompt has a maximum length of 48 characters. Note also that Prefix 8 will be left set to the target folder. This prefix will be used to set the dialog default the next time that Function 27 or 28 is used. The internal script Prefix 8 will not have been set by this call, so you should not depend on the numeric prefix '8:' as part of any pathname you use for normal script commands. Within the Browser's own Functions, just the filename on its own will be sufficient to retrieve the file after using this Function.

The Failed flag will be set if any error occurs or the Cancel button is pressed.

Function = 28 (PutFileName)

Function 28 returns a file name from a Standard Put File dialog:

```
EXT Browser 28 "PromptString" "FileName" <VarName>
EXT Browser 28 "Save which file?" "File" FileName
```

If the script command 'Put File' is used, the Browser will close and reopen round the call. This Function avoids that from happening.

Note that the Prompt has a maximum length of 24 characters, and the input FileName 31. Note also that Prefix 8 will be left set to the target folder. This prefix will be used to set the dialog default the next time that Function 27 or 28 is used. The internal script Prefix 8 will not have been set by this call, so you should not depend on the numeric prefix '8:' as part of any pathname you use for normal script commands. Normally just the filename on its own will be sufficient to save the file after using this Function.

The Failed flag will be set if any error occurs or the Cancel button is pressed.

Function = 29 (PrinterSetup)

Function 29 calls the standard printer Page Setup dialog:

```
EXT Browser 29
```

Function = 30 (PrintScreen)

Function 30 prints the Browser display through the standard print call:

```
EXT Browser 30 {"DocumentName"}
EXT Browser 30
```

If the optional "DocumentName" is given, this will be used for the print procedures.

Function = 31 (GetFormStatus)

Function 31 checks to see if a Form is currently active in the display:

```
EXT Browser 31
```

The Function will fail if a Form is not part of the current display, or any other error occurs.

Function = 32 (GetStyleCount)

Function 32 returns the number of actual style changes in the current display:

```
EXT Browser 32 <Count>
```

It appears that TextEdit will hang if it tries to open a Teach file with more than $3FF (1023) style changes. This function allows a script to check the number of style changes before it tries to save the display to a Teach file.

The Function will fail if any error occurs.

Function = 33 (GetLine)

Function 33 gets a line of data from the input stream into a variable:

```
EXT Browser 33 <TimeOut> <VarName1> {VarName2}
EXT Browser 33 10 String {EOLChar+$40}
```

This is similar to the 'Get Line' script command, but terminates input on either a CR, an LF or 256 characters. As the main script engine is not used, a TimeOut value in seconds must be given (1-65535).

VarName1 returns the string, and the optional VarName2 returns the EOLChar that was seen, or a null value, if 256 characters were reached. The EOLChar can only be a null, CR (ASCII 13) or LF (ASCII 10) and is returned as the ASCII value plus $40, to avoid scripting problems. Thus a CR would return as an “M”.

If an empty line is seen, it will keep looking till a complete line is seen, or the Timeout occurs. This allows for CR/LF pairs, where the first call gets the line to the CR, and then the second call would see the trailing LF as an empty line before the next line of data was received.

The Function will fail if any error occurs, or a Timeout happens before a complete line is seen.

Function = 34 (GetMenuStopID)

Function 34 gets the ID of a Menu Item to stop and show the page:

```
EXT Browser 34 <$MenuItemID>
EXT Browser 34 $$1234
```

The value must be given as a maximum 4 digit Hex value preceeded by a '$'. This will require two '$' characters in the script.

If the Menu Item with the given ID is hit, then if Bit 5 has been set, Function 14 will be effectively called to Show the Page. The 'Menu Hit' will still be reported if Function 2 is ON.

Disable the function by passing a null value of $0000, or by turning off Bit 5 of the BrowserPrefs. The value will be cleared when the controlling script stops.

The Function will fail if an invalid number is given.

Function = 35 (ReturnMetaStrings)

Function 35 returns three variables representing content strings of a META tag:

```
EXT Browser 35 <VarName1> <VarName2> <VarName3>
EXT Browser 35 CookieString ExpireString PragmaString
```

These will return the CONTENT string for:

```
<META HTTP-EQUIV"Set-Cookie" CONTENT=".....">
<META HTTP-EQUIV"Expires" CONTENT=".....">
<META HTTP-EQUIV"Pragma" CONTENT=".....">
```

If the tags have not been defined, empty strings will be returned.

The Function will fail if the three variables are not supplied.

Function = 36 (GetLine2)

Function 36 gets a line of data from the input stream into a variable:

```
EXT Browser 36 <TimeOut> <EOLChar> <VarName>
EXT Browser 36 10 13 String
```

This is similar to Function 33, but terminates input on the EOLChar being seen or 256 characters. As the main script engine is not used, a TimeOut value in seconds must be given (1-65535).

The given EOLChar may only be a CR (13) or LF (10), and if a CR is given, then LFs in the input stream will be ignored and not returned. Similarly, if the given EOLChar is an LF, then CRs are ignored and not returned.

If an empty line is seen, a null string will be returned in VarName.

The Function will fail if any error occurs, or a Timeout happens before a complete line is seen.

Function = 37 (BarberPole)

Function 37 allows manual manipulation of the Barber Pole:

```
EXT Browser 37 <Value> {PartialAmount} {FullAmount}
EXT Browser 37 0
EXT Browser 37 1
EXT Browser 37 2
EXT Browser 37 3 10 23456
```

Where Value:

0 = Clear display
1 = Opens blank display or clears display to empty box
2 = Displays the Barber Pole. Each time this is called, it increments the display
3 = Displays the Thermometer from the two supplied values.

This function will only display the Barber Pole in the Menu bar while the Browser window is open.

The Function will fail if an invalid value or required statement is missing.

Memory use and reported Errors

Most errors are reported back immediately to the script by setting the Failed flag.

While processing data, the Browser uses the Text Edit tool extensively, using TEInsert, TECopy, and TEPaste to manipulate the data. If any of these calls fail, it is not always possible to directly report to the script that there has been an error. In these cases, the required function will not take place, or in the case of drawing the page, it will prematurely abort.

Most Web pages should not cause any problems, but very large pages might not display on a IIgs with only the minimum amount of memory available.

Scripting Reference

The Browser will not really operate without a controlling script. The script handles the GETting of new pages, displaying many of the progress messages and POSTing replies. It is also up to the script to parse strings returned to it and make them into valid URL's as necessary.

The simplest approach for a script is to start the process by sending the first URL in a GET message to call a new page, and then call Function 1 to clear any existing displayed page and start capture. It can then issue the script command 'Suspend Script', and wait till the page has been displayed, or there is response from the user, or an idle period elapses. The script will then resume with the variables defined in Function 2 set to appropriate values. The Kind variable will indicate what kind of event has just happened, and the script can then act accordingly.

If control is passed back to the script through no activity or a timeout, you may decide to resume a Suspend Script once more, or take some other kind of action. Do whatever seems suitable under the circumstances. For instance to stop a page, you could send the following string to the Browser by the Display script command and then call Function 14:

```
Display "> <hr><h3>Transfer Interrupted!</h3>"
```

The script should parse any returned strings according to the Kind of hit, and call the next page through the GET routine once more.

In order to give more control of the Browser, it is suggested that a custom Menu Bar is built using the WindowMgr XCMD. Hits from this Menu Bar are reported in a similiar way to hits on the Browser window. The script should monitor the variables associated with those hits and respond appropriately to them.

Note that a display loaded from a standard Teach file will not have corresponding raw source text. Note also that if the file was originally saved from the Browser, and loaded again using Function 4, then any embedded links will still work as normal.

It is good practice to call Store Settings at the start of a script and Restore Settings before the script closes. You should also turn off auto-receive of B+ and Zmodem as these could interfere with the smooth operation of the display.

Embedded Sounds

The Browser accepts embedded sounds in two formats. The first speaks a line of text and requires the ByteWorks Talking Tools to be installed. The second plays a sound by name. The sound must be installed in the '*:System:Sounds' folder and the Sound CDev must be active. Note that the name must match exactly the name of the installed sound.

```
<object sis-speech="This string will be spoken"></object>
<object sis-sound="PlayThisSound"></object>
```

If either of these tags is found, an icon will be shown on screen. Play the sound by double-clicking the icon.

The 'sis-speech' tag can have the following attributes:

```
gender="male" or "female"
tone="bass" or "treble"
pitch="value" range 1-10
speed="value" range 1-10
```

The default without any corresponding attribute is for 'gender="male" tone="bass" pitch="6" speed="6".

Spectrum XCMD Technical Notes

Written by: Ewen Wannop
Amended by: Ewen Wannop

November 1994
March 1998

This technical note describes the ChatterBox XCMD

The ChatterBox XCMD

The ChatterBox XCMD provides an enhancement to the SHR Normal screen and ChatLine for use within RTCs. Windows are displayed with the contents of incoming messages appended to the contents of a TextEdit control. A LineEdit control allows replies to these messages to be entered. Buttons control the Sending of the message and the Closing of the window.

Calling 'EXT Chatterbox' with no function number will display a dialog which allows the manual setting of the ChatterBox prefs. Although this was present in v1.0 of ChatterBox, it was not documented in this file. In version of Spectrum v2.1 and later, Chatterbox will install a Menu item in the 'File Transfer' Menu item of the Settings Menu that gives access to this dialog.

Up to 10 windows may be active at any one time.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT ChatterBox 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Add Message)

Function 1 creates a ChatterBox window:

```
EXT ChatterBox 1  <ChatNum> “StringToAdd” {“WindowTitle”}
EXT ChatterBox 1 4323 “Hi Dave, this is Ewen!” {“Chat with D.Hecker”}
```

The ChatNumber is any arbitrary number (1-65535) provided by the script. A list of 10 chat windows is maintained. If the window already exists, the "StringToAdd" is appendedto that window's TEHandle.

If a “WindowTitle” is given, the Title of that window will be changed to the new string. The maximum length for the title is 31 characters. A space should be used to start and finish all titles.

If the window already exists, the fields are updated. If the window does not exist, it is created. It is created behind all other windows, and will not be visible if the Online display is open, till Function 3 is called.

If ten windows have already been created, the failed flag is set, and the command fails.

Function = 2 (Get Reply)

Function 2 returns the next reply:

EXT ChatterBox 2 VarName1 VarName2
EXT ChatterBox 2 4323 “Hi Ewen, how are you?”

If no replies are pending, then the Failed flag is set, otherwise VarName1 is set to the ChatNumber of the first entry in the reply pending buffer. VarName2 is set to the string to send of that entry. The entry is removed from the reply pending buffer.

Function = 3 (Activate ChatWindow)

Function 3 activates a Chat window from the list:

EXT ChatterBox 3 ChatNumber
EXT ChatterBox 3 4323

If the ChatWindow has not been created, then the Failed flag is set. Otherwise the Chat window will be brought to the front and made active.

Function = 4 (Close ChatWindow)

Function 4 close a Chat window and removes it from the list:

EXT ChatterBox 4 ChatNumber
EXT ChatterBox 4 4323

If ChatNumber=0 then all the windows in the window list will be closed. Otherwise the window for ChatNumber will be closed. If ChatNumber is not found then the Failed flag is set.

Pressing the “Close” button in a window will close the window and remove it from the list.

Function = 5 (Get ChatInfo)

Function 5 returns the status of the ChatterBox window referred to:

EXT ChatterBox 5 Index VarName1 {VarName2}
EXT ChatterBox 5 1 ChatNumber {“WindowTitle”}

Index to window ranges from 1-10

If Index=0 then VarName1 is set to the number of chat windows that are currently open and VarName2 (if supplied) is not changed.

Otherwise VarName1 is set to the chat number being conducted in window #Index and VarName2 (if supplied) is set to that window's title.

If window #Index does not exist then VarName1 is to set to “0”, VarName2 (if supplied) to "", and the Failed flag is set.

Function = 6 (Done Chatting)

Function 6 closes a Chat session:

EXT ChatterBox 6

All windows are closed and all memory used by them is disposed off. This call is also made automatically when a script closes, or Spectrum shuts down.

Function = 7 (Set Prefs)

Function 7 sets ChatterBox preferences:

EXT ChatterBox 7 <SendKey> {AutoHide} {ShowMessage}

All values, if supplied must be in the range 0-1. SendKey controls whether the CR key will fire the Send button. A value of 1 will activate the CR key press equivalent, this is the default value. AutoHide controls the response after the message has been sent. A value of 1 will automatically send the target window to the back after the message has been sent. The default value is 0. ShowMessage controls whether a window will be brought to the front when a message has been added to it. A value of 1 will bring the window to the front when a message has been added. The default value is 0.

Function = 8 (Get ChatInfo2)

Function 8 returns the status of the ChatterBox window referred to:

EXT ChatterBox 8 <ChatNumber> VarName1 {VarName2}
EXT ChatterBox 8 Index {"WindowTitle"}

If the #ChatNumber window exists, VarName1 is set to the Index of the #ChatNumber window, and VarName2 (if supplied) is set to that window's title.

If window #ChatNumber does not exist then neither Variable is changed and the Failed flag is set.

Function = 9 (Load Settings)

Function 9 loads the settings from a disk file:

EXT ChatterBox 9
EXT ChatterBox 9

If the file does not exist the failed flag will be set.

Function = 10 (Save Settings)

Function 10 saves the settings to a disk file:

EXT ChatterBox 10
EXT ChatterBox 10

If this file exists when Spectrum is next started and the ChatterBox XCMD loads, these settings will be used as the new default settings. The ChatBox.Prefs file can be found in the Add.Ons:XCMDs folder. You can delete this file to restore the original default settings.

The Failed flag will be set if it was not possible to save the settings to disk.

ChatterBox Explained

The ChatterBox XCMD adds a user interface to a custom script using the SHR Normal screen.

When an appropriate ChatterBox message arrives, the script will process this message and send it to the XCMD. The XCMD displays the message in a custom window. Up to ten windows may be active at any one time. The script controls the initial display of these windows, and the activation of any inactive windows as needed.

When a window is displayed, the user may compose a reply to the message showing in the window. These replies are added to the reply pending buffer when the “Send” button is pressed. If the message is not retrieved from this queue, and sent by the script, it will be lost.

Messages that have been “sent” are added to reply pending buffer. The script must retrieve the messages from this buffer using the GetReply command and process them as necessary.

All the text relating to a specific window will display within the TextEdit control. The user may therefore scroll through the entire session with that caller. Incoming messages are shown in Normal text and replies are shown in **Bold** text.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop
Updated by: Ewen Wannop

May 1995
July 24th 1996

This technical note describes the Database XCMD

The Database XCMD

The Database XCMD gives a script full control to a random access database file. This is suitable for the storage and retrieval of varying sized records such as those used in the message database of a BBS system.

Each entry in a database file consists of three fields. A numeric value that can range from 0-65535, an uncompressed data field and a second optionally compressed data field. There is no practical limit to the size of the two data fields. Typically a BBS might hold a message number in the value field, a message header in the uncompressed field, and the body of the message in the optionally compressed field.

There is no real limit to how these fields are used. The database could hold a set of pictures, text references or even code segments. The data can be saved and retrieved regardless of its content.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT Database 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (OpenDBaseFile)

Function 1 opens or creates a Database file:

```
EXT Database 1 {FileNum} ":Path:FileName" {VarName}
EXT Database 1 0 ":Hard:MessageFile"
EXT Database 1 ":Hard:MessageFile" Filenum
```

Where Filenum is in the range 0-9. If the optional Filenum is not given, then the next available Filenum will be allocated and VarName must be given. The Filenum allocated will be returned in VarName. If a Filenum cannot be allocated the Failed flag will be set.

If the file already exists, and is of the reserved filetype $52 auxiliary type $8081, it will be verified and opened. If it does not exist and the path is valid, an empty database file will be created and opened. The subsequent functions all require a file to be opened first before they can be used. All open files will be closed when the script stops.

If any error occurs, including a bad input value, the Failed flag will be set.

Function = 2 (CloseDBaseFile)

Function 2 closes an open Database file:

```
EXT Database 2 FileNum
EXT Database 2 0
```

If the file is not already open, then this function will do nothing.
If any error occurs, including a bad input value, the Failed flag will be set.

Function = 3 (GetDBaseInfo)

Function 3 returns the total number of entries in an open Database file:

```
EXT Database 3 FileNum VarName
EXT Database 3 0 Result
```

If any error occurs, including a bad input value, the Failed flag will be set.

Function = 4 (SetDBasePrefs)

Function 4 sets the state of the compression flag:

```
EXT Database 4 Value
EXT Database 4 1
```

Where Value = 0-1 {0=No 1=Yes}

If this flag is set, then the second data field will be compressed before saving. This flag is also respected when rebuilding the database with function 11. Once this flag has been set, it will remain in force until Spectrum is shut down, or the XCMD is unloaded completely. If the flag is not set, then the second data field will be saved uncompressed.

For compression to take place there must be an XCMD present that responds to the IPC requestes to compress the data. If no XCMD responds, the data will always be saved uncompressed.

The default is for compression to be OFF. Note that this will be the state at boot time. If it is changed, the new state will remain in force until changed, or Spectrum Quits.

If any error occurs, including a bad input value, the Failed flag will be set.

Function = 5 (SearchDBaseFile)

Function 5 searches an open file for a target string in its first data field:

```
EXT Database 5 FileNum StartRecord “String” [{VarName} or {EditorHandle}]
EXT Database 5 0 0 “Target” Result
```

If VarName is given it will be set to the number of the entry where the target string was found. Note that this returned number is for the logical entry in the file. If any records are deleted, the logical number of successive entries will be changed.

If an EditorHandle is given, it will contain a list of all the logical entries that match the target “String”. Each entry is ended by a Return. Either a VarName or an EditorHandle is required.

The search will start at the next logical entry defined by StartRecord (1-NumRecords). The search respects the setting of the CaseSensitive flag. To start with the first record, StartRecord should be set to 0. If the StartRecord is proceeded by a minus sign ‘-’, the search will start at the previous record to that

defined by StartRecord. In this case, to start with the last record, use Function 3 to get the total number of records and increase that number by one.

If the target string is not found, or any error occurs, including a bad input or start value, the Failed flag will be set.

Function = 6 (SearchDBaseFile2)

Function 6 searches an open file for a target value in the numeric value field:

```
EXT Database 6 FileNum StartRecord Value [{VarName} or {EditorHandle}]
EXT Database 6 0 0 1234 Result
```

If VarName is given it will be set to the number of the entry where the target value was found. Note that this returned number is for the logical entry in the file. If any records are deleted, the logical number of successive entries will be changed.

If an EditorHandle is given, it will contain a list of all the logical entries that match the target Value. Each entry is ended by a Return. Either a VarName or an EditorHandle is required.

The search will start at the next logical entry defined by StartRecord (1-NumRecords). The search respects the setting of the CaseSensitive flag. To start with the first record, StartRecord should be set to 0. If the StartRecord is proceeded by a minus sign '-', the search will start at the previous record to that defined by StartRecord. In this case, to start with the last record, use Function 3 to get the total number of records and increase that number by one.

If the target value is not found, or any error occurs, including a bad input or start value, the Failed flag will be set.

Function = 7 (LoadDBaseRecord)

Function 7 returns a record from an open file:

```
EXT Database 7 FileNum RecordNum EditorNumber1 EditorNumber2 {VarName}
EXT Database 7 0 1234 ~E01234 ~E04321 {Result}
```

If the logical RecordNum exists, the ScriptEditors will be returned with the contents of the two data fields. The second data field will be uncompressed. If the optional VarName is given it will be returned with the contents of the value field.

If the target record is not found, or any error occurs, including a bad input value, the Failed flag will be set.

Function = 8 (LoadDBaseRecord2)

Function 8 returns a record from an open file:

```
EXT Database 8 FileNum Value EditorNumber1 EditorNumber2 {VarName}
EXT Database 8 0 1234 ~E01234 ~E04321 {Result}
```

The file will be searched for an entry holding Value in the value field. If such a record is found, the ScriptEditors will be returned with the contents of the two data fields. The second data field will be uncompressed. If the optional VarName is given it will be returned with the logical number of that record.

If the target value is not found, or any error occurs, including a bad input value, the Failed flag will be set.

Function = 9 (SaveDBaseRecord)

Function 9 saves a record to an open file:

```
EXT Database 9 FileNum EditorNumber1 EditorNumber2 RecordNum {Value}
EXT Database 9 0 ~E01234 ~E04321 1234 {4321}
```

Saves the ScriptEditors to the two data fields, compressing the second editor if the compressed flag is set. The entry will be saved to the end of file (RecordNum=0), or to the logical target record if it exists. If the optional Value is given this value will be saved in the first value field.

Note that when saving to an existing record entry, the existing contents will be lost. If the new record is larger than the old, the entry will be added to the end of the file. In such a case the logical position of the entry will be preserved. Repeated saving of new data may in time can cause the file to be severely fragmented. Periodically clean the file by calling Function 11.

If the target value is not found, or any error occurs, including a bad input value, the Failed flag will be set.

Function = 10 (SaveDBaseRecord2)

Function 10 saves a new record value field to an existing record in an open file:

```
EXT Database 10 FileNum RecordNum NewValue
EXT Database 10 0 1234 4321
```

If the entry RecordNum is not found, or any error occurs, including a bad input value, the Failed flag will be set.

Function = 11 (DeleteDBaseRecord)

Function 10 deletes a record from an open file:

```
EXT Database 11 FileNum RecordNum {EndRecordNum}
EXT Database 11 0 1234 {2345}
```

If the target entry is found, then the internal file pointers will be changed to skip over that entry. This will result in a hole in the file. It is not possible to recover the deleted entry later. Periodically clean the file by calling Function 11.

If the optional EndRecordNum is given, all entries between RecordNum and EndRecordNum will be deleted.

If the target value is not found, or any error occurs, including a bad input value, the Failed flag will be set.

Function = 12 (RebuildDBaseFile)

Function 12 rebuilds an open file:

```
EXT Database 12 FileNum “:Path:FileName”
EXT Database 12 0 “:Hard:NewMessageFile”
```

Database files can become fragmented or contain holes where entries have been removed. Periodically they should be cleaned with this function. The records will be extracted from the open file, in their logical sequence and written out to a new Database file. An error will occur if you try to write to the current Database file. If the compression flag is set, and the field has not already been compressed, the

second field will be compressed if a compression XCMD responds to the IPC messages. If the compression flag is not set, the second field will be left in its current state.

If any error occurs, including a bad input value, the Failed flag will be set.

Function = 13 (MakeDBaseFile)

Function 13 converts a raw file into a Database file:

```
EXT Database 13 FileNum ":Path:FileName" ":Path:Template"
EXT Database 13 0 ":Hard:RawMessageFile" ":Hard:TemplateFile"
```

Raw files can result from a manual or automated online session with a Host system, or could be the result of repetitive data entered into a file from an appropriate source. Although this command was specially tailored for use with raw message files from online systems, it could be used with the output files of a spreadsheet or other data handling program.

The template file must conform to a fixed format, and must use <CRs> alone as the lineend marker. The information in that file is used to control the extraction of data from the source file, and thus the building of the various records in the output Database file. To denote control characters in the marker strings, use the standard Spectrum convention of a caret. Note that the Line End Character will only be seen once at the end of a marker string. The current setting of the Token Character is ignored here. If a line is empty, it will either be ignored, or the default value used. The template file must use <Returns> as end of line characters, any line feeds will be ignored.

Template file structure:

Line 1:	Line end character	Decimal Number	Default 13	0-255
Line 2:	Filter setting	Decimal Number	Default 0	0-2
Line 3:	Start file marker	String	Default empty	max 200
Line 4:	Record marker	String	Mandatory	max 200
Line 5:	Number lines first data field	Decimal Number	Default 2	0-255
Line 6:	Secondary record marker	String	Default empty	max 200
Line 7:	Number lines first fata field	Decimal Number	Default 2	0-255
Line 8:	End file marker	String	Default empty	max 200
Line 9:	Value field	Decimal Number	Default #0	0-65535
Line 10:	Flag	Decimal number	Default 0	0-3

Line 1 is used to denote the line end or entry separator in the source file.
Line 2 is a number (in ASCII):

- 0 All control characters are stripped.
- 1 All control characters are stripped, but if the Line end is ASCII 13, then tabs and linefeeds, other than as a Line end/Line feed pair, are turned into spaces.
- 2 All control characters characters are saved.

Line 3 is optional. If given records will not be stored till this sequence is seen.
Line 4 is mandatory, and is the marker that is used to separate records.
Line 5 is the number of lines, that will saved to the first data field. Everything from that point, till the next Record marker, will be saved to the second data field.
Line 6 is an optional secondary marker. This allows the distinct separating of subject and topic headers. If the Value field has been set as #0, then bit 15 will be set of the Value flag.
Line 7 is the number of lines for the secondary marker, that will saved to the first data field. Everything from that point, till the next Record marker, will be saved to the second data field.
Line 8 is optional. If given, it marks the sequence that stops the storing of records.

Line 9 is the value that will be used to fill the Value field. This will be used as the start value, and will be incremented for each record, wrapping when it gets to 65535. If the number is preceeded by a '#', then the number will be inserted into each value field, and will not be incremented. The default is for a '0' value. Use the HodgePodge Logical Operations to set the 8 bits of the Value.
Line 10 is a flag. If bit 0 = 1, then the marker strings from the first marker will not be included in the data fields, if bit 1 = 0, then the marker strings from the second marker will not be included. If bit 2 = 0 then data will be saved from immediately after the start marker is seen, it bit 2 = 1 then nothing will be saved till the record marker or secondary marker is seen.

Note that the end of line characters are saved to the record fields. The current CaseSensitive setting will be used on the marker searches.

If any error occurs, including a bad input value from the template file, the Failed flag will be set.

Function = 14 (LoadDBaseValue)

Function 14 returns the value field from a record in an open file:

```
EXT Database 14 FileNum RecordNum VarName
EXT Database 7 0 1234 Result
```

If the logical RecordNum exists, the VarName will be returned with the contents of the value field. This can be quicker to return the Value field than using Function 7. It also does not require the two ScriptEditors to be created.

If the target record is not found, or any error occurs, including a bad input value, the Failed flag will be set.

Compression

The DataBase XCMD does not have any compression routines of its own. When saving a record, or rebuilding a file, it will check the state of the compression flag. If this flag is set, it broadcasts an IPC message with StopAfterOne requesting any suitable XCMD to compress a ScriptEditor handle. If an XCMD responds, the compressed data is then saved to the file along with a marker returned by the compression XCMD. This marker will be used to flag for a suitable XCMD to decompress the data when subsequently retrieving the record.

The IPC messages are similar to the other IPC messages broadcast by Spectrum, this is an extract from the XCMD.Info file:

MESSAGE $8004 = Compression request sent to 'Spectrum XCMD~'

Compress:

DataIn word		3	number of entries
	word	1	info code 1=Compress, 2=DeCompress
	longword	$E01234	ScriptEditor handle
	word	0	method (0 = use best compression)
DataOut	word	x	1 = accepted
	longword	$E01234	new ScriptEditor handle
	word	x	compression method (0 = none)

DataIn+4 holds a TextEdit handle. DataIn+8 will normally be set to 0 (use best compression method). On return DataOut will indicate whether compression was made or not. To request a specific compression method, a specific value would be placed in DataIn+8.

If a module can compress the data, it should do so, and return a TextEdit handle in DataOut+2 with the compressed data and dispose of the original TextEdit handle. It should then place an identifying compression method flag into DataOut+6, and accept the message.

DeCompress:

DataIn	word	3	number of entries
	word	2	info code 1=Compress, 2=DeCompress
	longword	$E01234	ScriptEditor handle
	word	$EE	method
DataOut	word	x	1 = accepted
	longword	$E01234	new ScriptEditor handle
	word	0	not used

DataIn+4 holds a TextEdit handle, DataIn+8 holds an indentifying compression method flag. If a module identifies the method and can decompress the data, it should do so, and return a TextEdit handle in DataOut+2 with the decompressed data and dispose of the original TextEdit handle. It should then accept the message.

File Structure

The database files have type $52 and auxiliary type $8081. They consist of a special zero record which holds some global information, and then an array of variable sized records. Note that after considerable use the file may become fragmented with holes caused by deleted records, new records that are smaller than ones they are replacing, or by new records which being larger than the record they replaced, are saved at the end of the file. To find all the records in sequence follow the Next-Last pointers.

Zero record:

+0	$0000002B	Longword	Pointer to next record
+4	$00000000	Longword	Pointer to previous record
+8	$0000002B	Longword	Length of this entire record
+12	$00000017	Longword	Length of uncompressed first data field
+16	$0000	Word	Value field
+18	‘Database XCMD’		First data field (Unique file identifier)
+31	$xxxxxxxx	Longword	Total number of records
+35	$xxxxxxxx	Longword	Pointer to final record
+39	$00000000	Longword	Zero marker

Standard record:

+0	$xxxxxxxx	Longword	Pointer to next record
+4	$xxxxxxxx	Longword	Pointer to previous record
+8	$xxxxxxxx	Longword	Length of this entire record
+12	$xxxxxxxx	Longword	Length of uncompressed first data field
+16	$xxxx	Word	Value field
+18	Uncompressed Data		First data field
+xx	$xxxxxxxx	Longword	Length of compressed second data field
+xx	$xxxx	Word	Compression method
+xx	Compressed Data		Second data field

Spectrum XCMD Technical Notes

Written by: Ewen Wannop January 1995
Updated by: Ewen Wannop January 1995

This technical note describes the Debug XCMD

The Debug XCMD

The Debug XCMD provides a more graphic way of tracing script execution than using Set Debug Screen, Set Debug File or Set Debug Scrollback. It can be used in addition to all three of those commands to make script debugging a more pleasurable experience.

If used on its own, the Spectrum script command 'Set Debug External' will only display selected command lines within the Debug window. If the CarryOn flag has been set, the Set Debug Screen, Set Debug File or Set Debug Scrollback commands will be handled normally after the External call has been made. In addition, it is possible to control the handling of command lines manually to the Scrollback buffer by keypresses made when the Debug window is displayed.

The Debug window will display correctly on all SHR 640 screens and all 80 column Text screens. It will show oversized on SHR 320 screens.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT Debug 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Set Trap flag)

Function 1 sets a flag to control which script lines will be displayed:

```
EXT Debug 1 <Method>
EXT Debug 1 0
```

The<Method> indicates:

0	Both Script and XCMD command lines (Default)
1	Script command lines only
2	XCMD command lines only

The Failed flag is set if the value for Method is invalid.

Function = 2 (Set CarryOn flag)

Function 2 sets a flag which controls what Spectrum will do after the Debug call:

```
EXT Debug 2 <Method>
EXT Debug 2 0
```

The<Method> indicates:

0 Command lines will not be passed on to other Debug commands (Default)
1 Command lines will be passed on to other Debug commands

The Failed flag is set if the value for Method is invalid.

The Debug Display

The Debug Display shows three distinct pieces of information.

Title Line - Displays the Line number and Command number
The Line number is the current line within the script
The Command number is the current command within that line

Script Line - Displays the current 'raw' script line without substitutions
The line may be truncated to fit within a single line of the display

Debug or XCMD Line - Displays the processed line with its substitutions
The line may be truncated to fit within four lines of the display

Except for Tab characters, which are displayed as spaces, command characters are displayed in their '^A' format. Extended characters may display incorrectly on the Text screen as they are passed through a filter to remove the hibit when the text screen is being displayed.

While the Display is showing on the screen, the following key presses are active:

Escape	Calls Stop Script
OA-Period	Calls Stop Script
OA-R	Calls Stop Script
C or c	Turns on the CarryOn flag
X or x	Turns off the CarryOn flag
S or s	Calls Set Debug ScrollBack
O or o	Calls Set Debug Off and Set Debug External
W or w	Calls Set Debug ScrollBack and Clears ScrollBack

Any other key steps the debug process normally

Note that pressing 'O' will also call Set Debug External to turn the Debug XCMD back on. The Debug XCMD also swallows all the calls it generates by these key presses. They will not show on screen or in the Scrollback buffer. This is to aid the interpretation of the Debug record.

By using the 'S' and the 'O' keys, it is only necessary to put the single 'Set Debug External' command at the head of the script, or at the point you wish to start debugging from. You can then toggle the ScrollBack buffer on and off by these keys and view only the portion of script you are interested in.

Alternatively, if both Set Debug External and Set Debug Screen, Set Debug File or Set Debug Scrollback have been called, you can use the 'C' and 'X' keys to manually control the CarryOn flow of the Debug process.

Warning: As the Debug display 'swallows' key presses, you will not be able to use the Escape key in its normal mode. With the Debug XCMD active, and its display showing, pressing Escape will only stop a script, it will not call any On Escape commands that may be active.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop May 1995
Updated by: Ewen Wannop May 8th 1995

This technical note describes the Diary XCMD

The Diary XCMD

The Diary XCMD allows scripts to be run at set events, times or intervals. It adds greater control of timed events than the WaitFor Time command allows and adds more flexible control of startup scripts.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

EXT Diary 0 <VarName>

The VarName, if present, will return the version number of the XCMD.

Function = 1 (SetEvent)

Function 1 adds or resets an event in the event list:

EXT Diary 1 “EventName” “:Path:FileName” When “Time” {Priority} {Count}
EXT Diary 1 “Event_1” “:Hard:Script” 4 “95:05:19:23:12:00” {1} {2}

Where When:
1 = Script run at Startup
2 = Script run at FileXfer close
3 = Script run at specified date and time
4 = Script run hourly at minutes/seconds specified
5 = Script run daily at minutes/seconds specified
6 = Script run weekly at minutes/seconds specified
7 = Script run monthly at minutes/seconds specified
8 = Script run at ShutDown

Where Priority:
0 = Default - Runs when current script finishes
1 = Shuts down any running script immediately
2 = ‘BackLog’ the event

Where Count: 0 = Default - Forever
x = Number of times to repeat the event

“EventName” is a name of up to 16 characters which uniquely identifies this event.
“:Path:FileName” indicates the target script.
“Time” is the full date time in this format: “Year:Month:Day:Hour:Minutes:Seconds”. You can use the HodgePodge ‘Get Date-Time’ command to generate your input. Using the HodgePodge command, the format should be: “YY:MO:DD:HH:MM:SS”. Hours are in 24 hour format.
Note that some events may not use the entire Time string. All fields must be present however, but may be left as the null value ‘00’ where they will not be used.

Events are checked for and triggered by the IPC idle messages sent by Spectrum. They may not fire precisely on the target time as a result, but will fire as soon as an idle message is transmitted after that

event. Events will not be 'BackLogged' unless the BackLog flag is set. BackLogged events will run as soon after the scheduled time as is practicable, though this may mean the next time that Spectrum is run.

If any input error occurs the Failed flag will be set.

Function = 2 (ClearEvent)

Function 2 clears an event from the event list:

```
EXT Diary 2 "EventName"
EXT Diary 2 "Event_1"
```

If the event does not exist, or an input error occurs, the Failed flag will be set.

Function = 3 (ListEvents)

Function 3 lists the current events in a list dialog and allows the deletion and addition of specific events:

```
EXT Diary 3
```

If the SHR display is not showing, this function will do nothing.

Function = 4 (Override)

Function 4 overrides the current event handling:

```
EXT Diary 4 How
EXT Diary 4 0
```

Where:	How = 0	restores normal event handling
	How = 1	turns off event processing till further notice

If an input error occurs, the Failed flag will be set.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop April 1995
Updated by: Ewen Wannop April 1995

This technical note describes the Freezer XCMD v1.0

The Freezer XCMD

The Freezer XCMD is a brute force method to reduce interrupt problems when downloading files at high transfer speeds. In a IIgs without an accelerator, or a IIgs loaded with a number of interrupt intensive Inits or DAs, it is very easy to get corruption on file transfers by the simple movement of the mouse. The extra interrupts generated by the mouse can easily put the interrupt system into overload and lost characters or bad checksums will result.

In most cases it is only necessary for this XCMD to be present in the XCMD folder as it is fully controlled by the IPC messages sent from Spectrum when a file transfer is in progress. The mouse interrupts will be turned off, and the cursor hidden, whenever a transfer starts. The cursor is then restored to normal when the transfer stops. If a transfer is manually stopped by pressing the Escape key while at the SHR screen, the Quit Transfer dialog will show as before. The cursor however will reappear but will be frozen on screen. The buttons in this dialog respond to key presses, so you will need to select your choice by key presses rather than by the mouse.

If manual control of the Freezer XCMD is made from a script, then be aware that the cursor may remain frozen while the script is operating. You would not be able to navigate the menus by mouse in this situation. When the script stops, the cursor will restore to its normal operation. The only exception to this is the optional beachball cursor flag. If this is selected the flag will remain in force until the flag is turned off again or Spectrum quits.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT Freezer 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Increment Cursor Counter)

Function 1 increments the cursor counter:

```
EXT Freezer 1
EXT Freezer 1
```

If the override flag has not been set, this will result in the cursor being immediately frozen. The cursor will remain frozen until the cursor counter decrements to zero.

Function = 2 (Decrement Cursor Counter)

Function 2 decrements the cursor counter:

```
EXT Freezer 2
EXT Freezer 2
```

When the cursor counter has been decremented to zero, the cursor will unfreeze and display normally once more.

Function = 3 (Set Override Flag)

Function 3 sets the override flag:

```
EXT Freezer 3
EXT Freezer 3
```

When the override flag is set it will override the state of the cursor counter. If the cursor was currently frozen it will be immediately unfrozen and displayed.

Function = 4 (Clear Override Flag)

Function 4 clears the override flag:

```
EXT Freezer 4
EXT Freezer 4
```

When the override flag is cleared, the state of the cursor counter will take precedence once more. If the counter is not zero the cursor will be frozen again.

Function = 5 (Set Beachball Flag)

Function 5 sets the Beachball flag:

```
EXT Freezer 5
EXT Freezer 5
```

If the Beachball flag is set, whenever the cursor is frozen a rotating beachball will be displayed instead of the cursor being turned off. The mouse interrupts will remain off as before.

It will depend on the actual baud rate your are using, and the speed of your IIgs, whether this feature adds too much overhead to the system time. If you find that you are losing characters, or getting bad checksums, do not use this Function.

The Beachball sequence is displayed from a RunQ routine to avoid too great an interference with the system. This may result in a jerky effect to the spinning ball as interrupts are received.

Function = 6 (Clear Beachball Flag)

Function 6 clears the Beachball flag:

```
EXT Freezer 6
EXT Freezer 6
```

Clears the Beachball flag. If the cursor count is not currently zero, and the override flag is not set, the cursor will be frozen and hidden.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop — April 1995
Updated by: Ewen Wannop — October 2000

This technical note describes the HodgePodge XCMD

The HodgePodge XCMD

The HodgePodge XCMD is a miscellany of commands that cover many of the scripting requirements not covered within any of the othe XCMDs.

The original HodgePodge was written by Bill Tudor. The HodgePodge XCMD engine, and the first seven HodgePodge commands were written by Bill and survive intact. The further commands have been added to the HodgePodge XCMD by Ewen Wannop.

For versions of Spectrum later than v2.1, HodgePodge will add three items to the Extras menu. Calculate Checksums, Wrap MacBinary and UnwrapMacBinary. Calculate Checksums will prompt for the source file, and then display the results in an Alert. The MacBinary encoding will prompt for the source file, and then the destination file. If an error occurs in any of the processes, a generic Alert will be displayed.

HodgePodge v1.4 and later, support IXC commands. Refer to the main XCMD documentation on how to use these commands.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT HP 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (DeEmbed)

Function 1 removes embedded commands from a string:

```
EXT HP 1 “String” VarName1 VarName2 VarName3 VarName4
EXT HP 1 “This is a test.{s5 SystemBeep}” VarName1 VarName2 VarName3 VarName4
```

where: “String” = the string to parse (de-embed in this case)
VarName1 = the embedded command
VarName2 = String for the command
VarName3 = Number of times to repeat the command
VarName4 = Original string without the embedded command

An embedded command is encoded with an open brace “{” followed by the command character and optional repeat count, then a string for the command, ending with a close brace “}”. For example, “This is a test.{s5 SystemBeep} Did it work?”

The command character is “s”, the repeat count is 5, and the command string is “SystemBeep”. This embedded command plays sound. There can be many different types of embedded commands. The HodgePodge XCMD does not need to know anything about the command, it just parses the string.

In the above example, the four return variables would be:

VarName1 = “s”
VarName2 = “5”
VarName3 = “SystemBeep”
VarName4 = “This is a test. Did it work?”

If no embedded command is found, or if a syntax error exists in the string, the failed flag is set and the result variables are undefined.

Function = 2 (String2Text)

Function 2 converts an extended text string into ASCII text:

EXT HP 2 “String” TheResult
EXT HP 2 “This is a test of Spectrum™ ©1995” TheResult

This command converts a string which may contain any of the IIgs standard 8-bit characters (include trademark symbols, copyright symbols, etc.) into standard 7-bit ASCII text. The conversions are done via the StringToText toolbox call. The output text may be up to 4 times bigger than the input string (e.g., the “™” symbol can become “(TM” which is four characters instead of 1).

Function = 3 (ConvertVal)

Function 3 converts an Input value into different bases:

EXT HP 3 InputValue DecValue {HexValue} {BinValue} {BitValue}
EXT HP 3 170 170 $00AA 0000000010101010 0101010100000000

This command converts the “Input Value” into different bases. The input value must be in the range 0-65535 and be unsigned.

Input Value = a number to convert. A number begining with a "$" is taken as hexadecimal, one begining with a "%" is considered a binary number, and one begining with a "!" is considered a "Bits Value" (i.e., reverse binary - see below). Other numbers are considered decimal values. Negative numbers are not allowed.

DecValue = the value in decimal
HexValue = the value in hexadecimal
BinValue = the value in binary
BitsValue = a "Special" binary representation where the bits are in the reverse order (LSB is on the left), e.g., 3 = "10100000" vs "00000101"

Note: You do not have to include all of the result paramaters, however, you cannot "skip" parameters, i.e., if you want the binary value returned you must include the bits and hex return variables.

The failed flag will be set if any errors occur (including a bad input value).

Function = 4 (ConvertLong)

Function 4 converts a long Input value into different bases:

EXT HP 4 InputValue DecValue {HexValue} {BinValue} {BitValue}
EXT HP 4 2863311530 2863311530 $AAAAAAAA
1010101010101010101010101010101010
0101010101010101 0101010101010101

This command converts the “Input Value” into different bases. The input value must be in the range 0-4294967295 and be unsigned. See the discussion above for function number 3 for information regarding the parameters.

Function = 5 (ConvertValS)

Function 5 converts an Input value into different bases:

```
EXT HP 5 InputValue DecValue {HexValue} {BinValue} {BitValue}
EXT HP 5 -10 -10 $FFF6  1111111111110110 0110111111111111
```

This command converts the “Input Value” into different bases. It is the same as the ConvertVal Function except that values may be signed and must be in the range -32768 to 32767.

Function = 6 (ConvertLongS)

Function 6 converts a long Input value into different bases:

```
EXT HP 6 InputValue DecValue {HexValue} {BinValue} {BitValue}
EXT HP 6 -10 -10 $FFFFFFF6
        11111111111111111111111111110110
        01101111111111111111111111111111
```

This command converts the “Input Value” into different bases. It is the same as the ConvertLong Function except that values may be signed and must be in the range of -2147483648 to 2147483648.

Function = 7 (String2Text2)

Function 7 converts an extended text string in an Editor Handle into ASCII text:

```
EXT HP 7 “$EditorHandlex”
EXT HP 7 ~E01234
```

This command converts text in a script editor which may contain any of the IIgs standard 8-bit characters (include trademark symbols, copyright symbols, etc.) into standard 7-bit ASCII text. The conversions are done via the StringToText toolbox call. The output text may be up to 4 times bigger than the input string (e.g., the “™” symbol can become "(TM)", which is four characters instead of 1). The output text cannot exceed 64K.

Note that this command is very similar to the String2Text command (Function 2) except that it uses a ScriptEditor instead of a text string.

Function = 8 (MultiGet)

Function 8 retrieves one or more filenames into a ScriptEditor handle using a Standard File dialog. The entries are delimited by a CR:

```
EXT HP 8 “Prompt” Kind Type EditorHandle {CheckBox1} {CheckBox2} {CheckBox3}
EXT HP 8 “Get which file?” 0 0 ~E01234 3 {“Select Check One” 1 VarName}
```

Where a CheckBox entry is:

“Prompt” Value VarName (Value = 0 not checked, = 1 checked)

The Prompt and Checkbox strings are a string of up to 50 characters. This string will normally be displayed fully, but if all the characters in the string are wide, it is possible it will be truncated on screen. The failed flag will be set if this length is exceeded.

Where Kind: 0 = Any file
1 = Text file only (includes SRC files)
2 = Text, Teach, AppleWorks
3 = Launchable applications
4 = Any file without a resource fork

Where Type: 0 = No directories returned
1 = Directories returned - indicated by a trailing colon

Note that for Kind = 2 Teach files will be selectable. Teach files do have resource forks so this option is unsuitable for choosing files to be sent by protocol transfer. It is suitable for sending as text files.

Prefix 8 will be set to the directory that was chosen on return from this command.

The failed flag will be set if the Cancel button is pressed, the EditorHandle is not a valid handle and if any errors occur (including a bad input value). The EditorHandle will be empty if the Cancel button is pressed.

Function = 9 (SelectFolder)

Function 9 presents a Standard File dialog to select a folder. Optionally the selected foldername can be returned in a variable. The user should press Select while within the target folder. On exit Prefix 8 will be set to the folder path:

EXT HP 9 “Prompt” {VarName}
EXT HP 9 “Select in which folder?” {FolderName}

The Prompt string is a string of up to 50 characters. This string will normally be displayed fully, but if all the characters in the string are wide, it is possible it will be truncated on screen. The failed flag will be set if this length is exceeded.

Prefix 8 will be set to the directory that was chosen on return from this command.

The failed flag will be set if the Cancel button is pressed or if any errors occur (including a bad input value).

Function = 10 (FindFile)

Function 10 searches from a specified folder and finds files compared against a match criteria within that folder and any sub folders. It returns a list of files within a supplied ScriptEditor:

EXT HP 10 “:PathName:” SearchDepth EditorHandle {“MatchString”} {MatchSpec}
EXT HP 10 “Hard.Disk:Spectrum:” 1 ~E01234 {“Spectrum”} {0}

Where SearchDepth: 0 = Specified folder only
1 = Specified folder and sub folders

Where MatchSpec: 0 = Exact Match to MatchString (default if not supplied)
1 = Contains MatchString
2 = Begins with MatchString
3 = Ends with MatchString
4 = Length is less than MatchString (decimal value)
5 = Length is equal to MatchString (decimal value)
6 = Length is greater or equal to MatchString (decimal value)
7 = File Attributes equal MatchString
8 = FileType/AuxType equal MatchString

9 = Creation date/time equals MatchString
10 = Modification date/time equals MatchString

Notes:
(7) File atttributes are the same as those returned by 'Get FileInfo'. These are 'DNWBR', and MatchString must therefore be five characters long. Use a period '.' for OFF, and the appropriate character for ON. Use an 'X' for 'Don't care'.
(8) FileType and AuxType are the same as used by 'Set FileInfo'. The string must be seven characters long and there must be a '/' in the third position. Values are in Hexadecimal.
(9 & 10) The data is the same as returned by 'Set FileInfo', and must be either six or ten characters long. The format is either 'YYMMDD' or 'YYMMDDHHMM'.

If the optional MatchString is not supplied, then all files will be returned.

The failed flag will be set if the specfied folder does not exists or any errors occur (including a bad input value).

Function = 11 (MacBinary)

Function 11 allows files to be encoded with a MacBinary header, or be decoded back to the original file:

EXT HP 11 ":PathName:FileName" Kind VarName
EXT HP 11 "Hard.Disk:Spectrum:Spectrum" 1 VarName

Where Kind: 0 = Decode MacBinary
1 = Encode MacBinary
2 = Smart MacBinary toggles the state of file

HodgePodge will attempt to name the encoded file with the suffix '.bin' and a decoded file to its original name. If this is not possible, a limited level of renaming will be done. If the file still cannot be saved after being renamed, the failed flag will be set. The saved file is always placed in the same folder as the original file so you must allow at least the same space as the source file for the encoding/decoding process. The original file is not erased after the process is finished. VarName will be set to the name of the converted file if the conversion was successful.

If the file already is MacBinary encoded when Kind =1, or has no MacBinary header when Kind = 0, the function will return with VarName set to null and the failed flag cleared. You should check the state of VarName to see if the file was encoded/decoded. When Kind = 2, the file will be decoeed if it has a MacBinary header or encoded if it has not.

The failed flag will be set if any errors occur (including a bad input value).

Function = 12 (GetDateTime)

Function 12 allows the user to enter a date and time in an interactive dialog:

EXT HP 12 "TitleString" "Prompt string" VarName
EXT HP 12 "Please enter the date:" "YYMODD" VarName
EXT HP 12 "Please enter the date:" "The year is 19YY" VarName
EXT HP 12 "Please enter the date:" "The year is YYYY" VarName

The TitleString will be displayed at the top of the dialog. It is truncated to 44 characters, though care should be taken that the supplied string actually fits the window.

The date and time display will be primed with the current time when the dialog opens. The small black pointer arrows will change the entries. First click on the entry you wish to change. On pressing the OK button the supplied prompt string will be returned in VarName. Any occurences of the following letter pairs in the supplied string will be replaced by their corresponding date/time values. This allows great flexibility as the input string can contain these pairs embedded at random within normal text. Note that the letter pairs must be in upper case:

YYYY = Long years YY = Years MO = Months
DD = Days HH = Hours MM = Minutes
SS = Seconds

The failed flag will be set if the Escape key is pressed or if any errors occur (including a bad input value).

Function = 13 (DaysToDates)

Function 13 calculates the number of days between two dates, and returns the result in a variable:

EXT HP 13 “YYMMDD” “YYMMDD” VarName
EXT HP 13 “$Year$Month$Day” “$Year$Month$Day” VarName
EXT HP 13 “950423” “950424” Result

The entry dates must be in the format shown. It does not matter which way round the two dates are, the correct number of days will be calculated.

The failed flag will be set if any errors occur (including a bad input value).

Function = 14 (TextEditor)

Function 14 displays the contents of a ScriptEditor handle in an editable window. On exit the ScriptEditor handle will contain the contents of the edited record:

EXT HP 14 “Title” EditorHandle {Left Right Top Bottom}
EXT HP 14 “Edit Record” ~E01234 3

The “Title” has a maximum width of 32 characters. If the optional position is not given, the display will be set to full screen, otherwise it sets the size and position of the window. The window sizes are within the following limits:

Left = 0-505
Right = 135-640
Top = 26-150
Bottom = 76-200
Minimum width = 135
Minimum height = 50

While this window is active, you will not be able to select any other menus or windows. The window can be sized, moved, zoomed and otherwise handled normally.

The failed flag will be set the EditorHandle is not a valid handle and if any errors occur (including a bad input value).

Function = 15 (CalcCheckSum)

Function 15 calculates a 32 bit checksum for the data and resource forks of a file. It returns the values as two Hex values in two variables:

EXT HP 15 ":PathName:FileName" DataForkCRC {ResForkCRC}
EXT HP 15 "Hard.Disk:Spectrum:Spectrum" DataForkCRC ResForkCRC

results in:

DataForkCRC = '$CF681AAF'
ResForkCRC = '$808162F3'

The second variable for the resource fork CRC is optional. If either fork is empty, the returned Hex CRC value will be $00000000.

Note that the CRC returned by this function is the same as a CRC calculated by a Zmodem transfer. There is no standard for CRC calculations and they can be generated in many different ways using different polynomial bases and seed values. The CRC's generated by HodgePodge are suitable only for checking against another CRC also generated by HodgePodge. They do not return the same values as the 'Calc CRCs' desk accessory by Harold Hislop. In practice this would not be a problem, as most uses of this function would be to check that two files were the same or different.

Although this function is fast in operation, do allow time for a large file to be checked, especially if you are reading this file froma a floppy disk or slow device. If you need the output in some other format than a Hex string, you can convert it to another format by passing the value you obtain back through function 6 of HodgePodge.

The failed flag will be set if any errors occur (including a bad input value).

Function = 16 (LogicalOperation)

Function 16 performs a logical operation on two supplied Hexadecimal values:

EXT HP 16 "FirstValue" "Operator" "SecondValue" VarName
EXT HP 16 "$10101010" "EOR" "0000FFFF" Result

Valid operators are: AND
ORA
EOR

The FirstValue will be operated on by the SecondValue. Thus in the example shown the Result will be "1010EFEF". Use HP Functions 3-6 to generate Hexadecimal values for input to this command. Note that a '$' is optional and is implied. The returned VarName will contain a Hexadecimal value without a '$'.

Note that if the logical operations ROR, ROL, LSR and ASL are required you can first convert the source value to a BinValue using Functions 3-6 and then manipulate the BinValue using normal string operations.

The failed flag will be set if the input values do not equate to a Hexadecimal value or if any errors occur (including a bad input value).

Function = 17 (ChooseFont)

Function 17 presents the standard ChooseFont dialog with the current Font selected:

EXT HP 17 VarName1 {VarName2} {VarName3}
EXT HP 17 FontID {Size} {Style}

Sample return: Variable1 = $88EA
Variable2 = 8
Variable3 = 0

The first variable will return the Hexadecimal value of the selected Font. The optional second variable returns the decimal size, and the optional third variable returns the style.

The failed flag will be set if the dialog is cancelled and if any errors occur (including a bad input value).

Function = 18 (CheckSum1)

Function 18 calculates a 32 bit checksum for the supplied string. It returns the values as a Hex value in the variable:

EXT HP 18 “String” StringCRC
EXT HP 18 “This is a supplied String” StringCRC

results in:

StringCRC = ‘$CF681AAF’

If string is empty, the returned Hex CRC value will be $00000000.

Function = 19 (CheckSum2)

Function 19 calculates a 32 bit checksum for the supplied Handle. It returns the values as a Hex value in the variable:

EXT HP 19 ~EditorHandle HandleCRC
EXT HP 19 ~E01234 HandleCRC

results in:

HandleCRC = ‘$CF681AAF’

If Handle is empty, the returned Hex CRC value will be $00000000.

Function = 20 (DaysToDates2)

Function 20 calculates the number of days between two dates, and returns the result in a variable:

EXT HP 20 “YYYYMMDD” “YYYYMMDD” VarName
EXT HP 20 “$LongYear$Month$Day” “$LongYear$Month$Day” VarName
EXT HP 20 “19950423” “20050424” Result

The entry dates must be in the format shown. It does not matter which way round the two dates are, the correct number of days will be calculated. This Function is Year 2000 compliant and correctly calculates dates in the new Millenium.

The failed flag will be set if any errors occur (including a bad input value).

Function = 21 (TimeToSecs)

Function 21 calculates the number of seconds represented by a given date, and returns the result in a variable:

EXT HP 21 “YYYYMMDDHHMMSS” Format VarName
EXT HP 21 “$LongYear$MonthDayHour$Minute$Second” 1 VarName
EXT HP 21 “19950423123654” 1 Result

The Format value:
0 = Mac format, seconds from 1st Jan 1904
1 = Unix format, seconds from 1st Jan 1970

Dates are valid from midnight on the 1st January 1904 until 28:15 on the 6th February 2040. The failed flag will be set if any errors occur (including a bad input value).

Function = 22 (SecsToTime)

Function 21 calculates the date represented by a seconds value, and returns the result in a variable:

EXT HP 22 “Seconds” Format VarName
EXT HP 22 “123456789” 1 VarName

The Format value:
0 = Mac format, seconds from 1st Jan 1904
1 = Unix format, seconds from 1st Jan 1970

Result VarName:
“YYYYMMDDHHMMSS”

Dates are valid from midnight on the 1st January 1904 until 28:15 on the 6th February 2040. The failed flag will be set if any errors occur (including a bad input value).

Function = 23 (getForkSizes)

Function 23 returns the sizes of the two forks in decimal values:

EXT HP 23 “:Path:Filename” DataFork {ResourceFork}
EXT HP 23 “:Hard.Disk:File” Data Resource

The returned values are in decimal. This is a useful adjunct to the Get FileInfo script command, as it can be used to determine if a file has a resource fork or not. A file without a fork will always return a zero length for the resource, whereas a file with an empty resource fork, will always return a finite size.

The failed flag will be set if any errors occur.

Function = 24 (getTick)

Function 24 returns the current value of the system Tick counter:

EXT HP 24 VarName

The returned value is in decimal. This is a useful function to allow scripts to control the double click of a List control in a window built using the WindowMgr XCMD.

The failed flag will be set if any errors occur.

Function = 25 (getVolumes)

Function 25 returns information about all volumes currently active:

EXT HP 25 ~EditorHandle

The EditorHandle will return the information as a series of CR delimited lines. If a device has no volume mounted, only the device name will be returned and the rest of the string will be blank:

Start	*Length*	*Contents*
1	31	Device Name
33	31	Volume Name
65	10	Volume size in bytes
76	10	Volume free in bytes
87	2	FileSystem ID

FileSystem ID

1	ProDOS/SOS	2	DOS 3.3
3	DOS 3.2 or 3.1	4	Apple II Pascal
5	Macintosh (MSF)	6	Macintosh (HFS)
7	Lisa	8	Apple CP/M
10	MS/DOS	11	High Sierra
12	ISO 9660	13	AppleShare

The failed flag will be set if any errors occur.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop January 1995
Updated by: Ewen Wannop October 1998

This technical note describes the Kermit XCMD

The Kermit XCMD

The Kermit XCMD adds Kermit file transfer capability to Spectrum. The XCMD is controlled entirely from a host script and is able to support Kermit transfers from any of the standard SHR or 80 column Spectrum display options.

An important new feature is the ability to transfer files with both data and resource forks using a MacBinary wrapper. Files downloaded by Spectrum have always been able to unpack these files, now you can send them as well.

The Kermit XCMD file consists of two code segments. The first is a normal static segment and the second a dynamic segment. To save memory, the larger dynamic segment is only loaded into memory when a file is being received or sent. It then remains active in memory till it is unloaded by the script stopping, or by manually unloading the segment by calling Function 6.

If the dynamic segment cannot be loaded for any reason, the Failed flag will be set whenever Function 1 or 2 is called. The failed flag will also be set if any other error is generated by sending or receiving a file. The last error that was generated can be retrieved by calling Function 7.

In addition to the memory used by the small XCMD, Kermit requires a 64K block of free memory to load the dynamic segment into. It is able to transfer files in both a 7 bit and 8 bit environment, and as Kermit only uses one control character ($01), it can also operate in network environments that are normally hostile to the usual X-Zmodem file transfer methods.

Kermit v2.0 onwards is Spectrum v2.1 compatible and will insert menu items into the Spectrum menus. This allows direct control of the Send, Receive and Settings functions from the Spectrum menus without the need of entering XCMD script functions. It is also aware of the Send and Receive default paths, and it will use these paths as appropriate. It also recognises and uses the default FileType and AuxType settings and supports the expanded protocol status display.

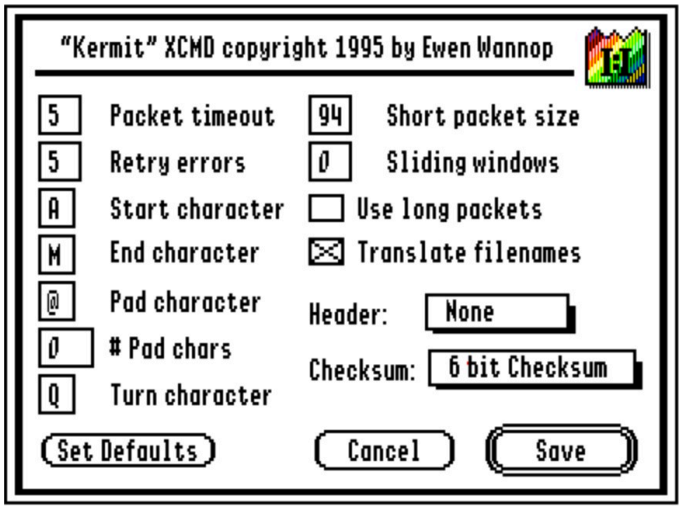
Kermit v2.0 onwards will not load under Spectrum 2.0, it requires Spectrum 2.1 or later. Kermit v2.4 onwards will not load under Spectrum 2.1, it requires Spectrum 2.2 or later.

Function = Null (Set Parameter dialog)

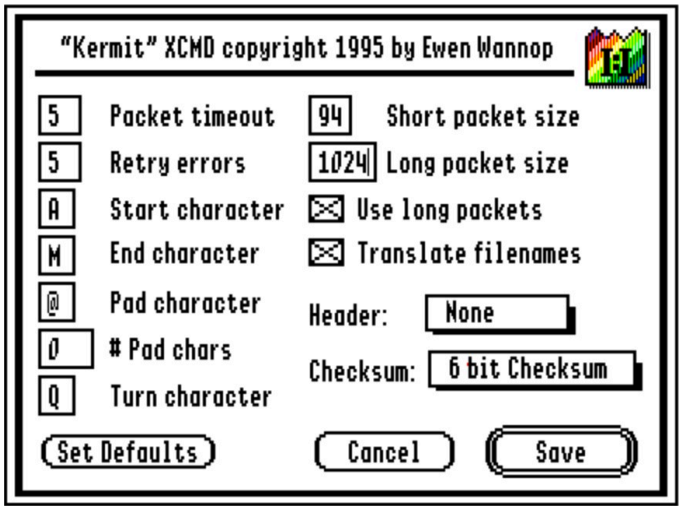
With no function specified, and with the SHR 640 desktop showing, a dialog window will open allowing you to manually set the various Kermit user adjustable parameters. See Function 5 for further details on these parameters.

```
EXT Kermit
```

The OK button will save the settings in the file ‘Kermit.Prefs’ in the Add.Ons folder. If this file exists when Kermit is next loaded, these saved values are used as the new defaults. Delete the file ‘Kermit.Prefs’ if you wish to restore the factory default parameters.



This shows the actual Kermit dialog. If Long packets have been selected, the Sliding windows box will be hidden, and the Long Packet size box will be shown instead:



Most items in this dialog are self explanatory, and are standard Kermit parameters found in most Kermit implementations.

Those settings needing explanation are:

- Use long packets - When selected will allow larger packets if the host supports this feature.
- Translate Filenames - When sending files, filenames will be converted to standard ASCII format. If you know that the host can accept filenames in native format do not check this box.
- Header - There are four settings for this popup. If a header has been selected, it will control the format of the file being sent or whether any embedded wrappers will be stripped on receiving files. See Function 5 for further details.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

EXT Kermit 0 <VarName>

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Sends a single or batch files)

Function 1 sends a single or batch files:

EXT Kermit 1 {Prefix} “FileName1” {“FileName2”} etc.
EXT Kermit 1 {Prefix} {EditorHandle}
EXT Kermit 1 10 “File1” {“File2”} etc.
EXT Kermit 1 “:Ram5:” “File1” {“File2”} etc.
EXT Kermit 1 “File1” {“File2”} etc.
EXT Kermit 1 ~E01234

If the optional Prefix is present the value of that prefix will be used to set Prefix 8 before each file is loaded. If a prefix is not present, or the filename does not have a prefix as part of its name, the current FileXferPath will be used instead. By adding a prefix as part of the filename, any other paths will be ignored. The Prefix can be a number, or a path enclosed by colons. If the Prefix is a number, the script should first set this prefix by using:

‘Set GSPrefix <PrefixNumber> “FolderName”’

before making this XCMD call. Any prefix may be used that is not reserved. Spectrum does not directly use any of the prefixes above 10.

If the optional ScriptEditorHandle is present, the command will use this as the source of FileNames. The Handle is checked to see if it is valid and the Failed flag will be set if it does not exist. ScriptEditorHandles are in the form of ‘~E01234’. Use the script substitution ‘$EditorHandle#’ to retrieve a ScriptEditorHandle as input to this command. If a ScriptEditorHandle is given, the rest of the command line will be ignored. If the optional Prefix is present, this will be added to the filenames extracted from the EditorHandle.

FileNames within an EditorHandle must be bracketed by the current QuoteCharacter if they contain any spaces. If they are not bracketed, the FileNames will be delimited by the first space character or carriage return. Carriage returns and linefeeds may be freely used between entries.

If the optional EditorHandle is not given, the FileNames listed on the command line will be sent. As many files can be sent as will fit into the EXT script command.

Note: The current Quote character **must** be used to enclose file and pathnames.

The Failed flag is set if the active segment cannot be loaded through insufficient memory.

Function = 2 (Receive a single or batch files)

Function 2 receives multiple files from a host:

EXT Kermit 2 {Prefix}
EXT Kermit 2 {Prefix}
EXT Kermit 2 10.
EXT Kermit 2 “:Ram5:”
EXT Kermit 2

If the optional Prefix is present it will be used as the directory for receiving files. If the Prefix is not present, the current FileXferPath will be used. The Prefix can be a number, or a path enclosed by colons. If the Prefix is a number, the script should first set this prefix by using ‘Set GSPrefix <PrefixNumber> “FolderName”’ before making the XCMD call. Any prefix may be used that is not reserved. Spectrum does not directly use any of the prefixes above 10.

Note that you will usually have to trigger the start of a single or batch Kermit transfer from the host before calling this Function.

Note: The current Quote character **must** be used to enclose file and pathnames.

The Failed flag is set if the active segment cannot be loaded through insufficient memory.

Function = 3 (Send Multiple Files)

Function 3 allows manual selection of files for batch sending:

EXT Kermit 3

A standard _SFMultiGet2 dialog will be shown. Multiple files can be selected from this dialog.
If the SHR desktop is not showing when this call is made, the Failed flag will be set.

Function = 4 (Set User adjustable Parameters)

Function 4 changes the Kermit default parameters:

EXT Kermit 4	A	B	C	D	E	F	G	H	I	J	K	L	M	N
EXT Kermit 4	7	5	1	A	M	@	0	Q	94	0	0	1024	1	0

The default settings are as follows:

A	Timeout in seconds:	5	1-63
B	Packet retries:	5	1-20
C	Error checking method:	1	1 = 1 byte Checksum 2 = 2 byte Checksum 3 = 16 bit CRC
D	Start of Packet Control Char:	A	Control ()
E	End of Packet Control Char:	M	Control ()
F	Padding Control Char:	@	Control ()
G	Number of Pad Characters:	0	0-255
H	Turn round Control Char:	Q	Control () (Xon)
I	Short Packet Length:	94	10-94
J	Sliding window count:	0	0-31 0 = no windows
K	Use Long Packets	0	0 = Short Packets & Sliding Windows 1 = Long Packets
L	Long Packet Length:	1024	10-9024
M	File Name Translation	1	1 = standard translation 2 = no translation
N	The transfer of attributes method:		0-3

Sending files:

0 = No wrapper added	
1 = Binary II wrapper	(no forked files allowed)
2 = MacBinary wrapper	(all files allowed)
3 = Smart wrapper added:	Text Files no wrapper Binary files data fork only = Binary II wrapper Binary Files with resource fork = MacBinary wrapper

Note: A Binary II or MacBinary wrapper will only be added if one does not already exist.

Receiving files:

0 = No wrapper stripped
1 = Binary II wrapper stripped
2 = MacBinary wrapper stripped
3 = Smart wrapper all files stripped

Not all parameters need be specified. The minimum is one. The failed flag will be set if any of the parameters are not within range.

Note that Long Packets and Sliding Windows are mutually exclusive. You can have one or the other. This is standard practice in Kermit implementations. If the connection is good enough to allow long packets, you do not need sliding windows and vice-versa.

The settings made by this function are temporary and are only held in memory while Spectrum is active. If you wish to save these values to disk, for alter use, you must also call Function 8.

Function = 5 (Shut down)

Function 6 shuts down the active module:

```
EXT Kermit 5
```

When a file transfer is requested, a dynamic segment of code holding the main code of the Kermit XCMD is loaded into memory. This segment will be unloaded whenever the script stops to save memory. It can be unloaded manually by Function 5 in order to save memory before a script stops.

Function = 6 (Get Error)

Function 6 returns the last error that was generated by the XCMD from a failure of either sending or receiving a file. An error is also generated if the dynamic segment could not be loaded. No error is generated when the Failed flag is set by an incorrect parameter. This function will return a value of '0' if a successful file transfer has been made.

```
EXT Kermit 6 <Method> <VarName>
EXT Kermit 6 0 ErrorResult
Display "$ErrorResult"
```

The<Method> indicates:

0 decimal number result
1 4 digit hexadecimal result in the format $0000

The Error number returned will be as follows:

Error = 0	No Error
GS/OS errors returned unchanged	Range $00-$7F
Memory errors returned unchanged	Range $0100-$FFFF
Kermit Errors:	Range $80-$8F

Refer to Appendix B for a full list of the Kermit errors that may be returned.

Function = 7 (Load settings)

Function 8 loads the 'Kermit.Prefs' file from disk replacing the current settings.
If the file does not exist the current settings are left unchanged.

```
EXT Kermit 7
```

Function = 8 (Save settings)

Function 8 saves the current settings to the file 'Kermit.Prefs' in the Add.Ons folder. If this file exists when Kermit is next loaded, these saved values are used as the new defaults.

```
EXT Kermit 8
```

Function = 9 (Temporary Debug mode)

Function 9 enters the Kermit Debug mode:

```
EXT Kermit 9
```

If the SHR 640 desktop is showing when this command is issued, you will be presented with a standard file dialog asking where you want the debug file to be placed. If you select a file to save to, all active packets being sent and received, will be saved to this file. Note that the file will be deleted if it already exists, so be careful not to point the dialog at any existing file you may still want! Make sure that you have sufficient space on the target disk or your debug file will be trubcated when the disk is full. It is best if you save this file to a Ram disk.

Once you have selected Function 9, it will keep saving a Debug file on every transfer or batch transfer, until the Kermit.Seg file is unloaded from memory.

The file has the following structure:

```
<CR>R:{Received Packet: [Header][Data][Checksum]}
<CR>S:{Sent Packet: [Header][Data][Checksum]}
```

As all we are usually interested in when we are debugging Kermit are the headers of the packets, and also the intialising transport packets, any packet that exceeds 32 bytes will be truncated with a $FF placed at its end. This length includes the <CR>R: or <CR>S: marker bytes. It is much easier to analyse a file which has short packets like this!

You may use the Debug feature to debug Kermit transfers yourself, or we may ask you to send us a Debug file if you are having problems with Kermit transfers.

This feature will probably be removed in the final release version.

Function = 10 (FreezeCursor)

Function 10 handles freezing of the cursor:

```
EXT Kermit 10 <value>
```

Where Value = 0 to restore the normal cursor, and Value = 1 to hide it. This function is provided for script control of the cursor during times when it could cause character loss if the cursor was displayed. It is the same process as used during downloads.

The cursor will be restored to normal when a script stops.

Appendix A

General Notes:

Both ends of a Kermit transfer will negotiate the transfer parameters before a Kermit session. The best settings will be kept if both ends support them. Individual settings will fall back to a common default if either end cannot support that particular setting.

Within the Status display, the CPS rating is calculated using the actual transferred data. The transmitted/received file size is calculated using the actual file data. This method is used to try and stablises what otherwise would be a widely fluctuating CPS rating due to the high level of prefixed characters in a Kermit transfer.

Packet length should be chosen to suit the integrity of your connection. If you have a bad line, choose smaller packets. Large packets can be used if the connection is good.

Kermit is transparent to both 7 and 8 bit connections. Where possible, when transferring Binary files, you should use an 8 bit connection to keep transfer times short.

In general you will find transfers are usually quicker if you do not use the Sliding Windows feature. If your network is slow, or prone to stuttering, try using Sliding Windows and see what kind of response you get. Use a small Sliding Windows value of around 9-10 at first. Larger values do not necessarily mean that the transfer will be smoother and faster. You may well need to experiment to find the optimum settings. As sliding windows does not use large packets, the speed of transfers will be limited.

If you choose to use larger packets, you may also find that the largest size does not necessarily mean a faster transfer. If you are experiencing noise on the line, a smaller packet size will probably be more efficient. A clean line will allow the use of the largest packet size.

You should not need to change any of the other settings unless your host is non-standard in its configuration. If you are having problems with a transfer, please check with the administrator on your host that you are using the correct settings.

Appendix B

Kermit XCMD Errors:

$00	No error
$80	Manually aborted transfer
$81	Timed out waiting for data
$82	Too many packet retries - error count exceeded
$83	Bad checksums on received packets
$84	Fatal error packet received
$85	Bad file type (resource fork without SmartSend/MacBinary)
$86	Bad input file name
$87	TCP/IP was not online

These errors are returned by Function 6. The error flag is reset on each call so it will only be valid if read immediately after a call. The Failed flag is also set when an error has been generated.

Appendix C

Menu insertion:

Kermit attempts to insert Menu items into Spectrum's menus when it is started up. Only Spectrum v2.1 and onwards will recognise the IPC messages and respond. Kermit v1.1 or later is fully compatible with Spectrum 2.0, but without the menu control.

Kermit Send and Receive items are placed in the Send and Receive File hierarchical menus, and a Kermit item is placed in the Settings menu.

When a menu hit is made on an inserted Kermit menu item, the XCMD will monitor the hits and call the Kermit functions directly. If an error occurs during the transfer the error is displayed in an Alert. The error is also stored and can be retrieved by Function 6.

For greater control of Kermit then the menu selection allows, use the normal XCMD script functions.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop March 1995
Updated by: Ewen Wannop March 1995

This technical note describes the Library XCMD v1.0

The Library XCMD

The Library XCMD provides a method for scripts to access an additional script file holding a library of common routines. This can save considerable space within a script by avoiding repetitive sequences. It also will save time in script development by using an existing library of appropriate routines.

The library files follow the standard script format and can contain most script commands.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT Library 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Load File)

Function 1 loads a library file into memory:

```
EXT Library 1  <PathName/FileName>
EXT Library 1 ":PathName:FileName"
```

The library script file must be a Teach or Text file and must not exceed 64K in length. It is constructed in the same way as a normal script file.

The Failed flag is set if the file is not found, the file is not a Teach or Text file, or it cannot be loaded through insufficient memory.

Function = 2 (Unload File)

Function 2 unloads the library script file:

```
EXT Library 2
EXT Library 2
```

The file will be unloaded clearing any memory that it took. The file is unloaded automatically when a script stops or Spectrum shuts down.

Function = Label

If no numeric command is given, and a name starting with an alpha character is found, then the name is assumed to be a reference to a standard Label within the library file.

When called by the following command:

Ext Library “RoutineName”

or:

Ext Library RoutineName

The label ‘RoutineName’ will be searched for. If the label not found the Failed flag will be set.

All script commands within the boundary formed by the target label at the start, the next blank line, empty line or the end of file as the end, will be passed back to Spectrum for execution.

Most script commands can be used within the library files. The most notable exception is that For Next loops cannot be used, as they involve a backward step in the script flow. GoSub or GoTo commands can only be used if they reference labels outside the library file. In this case, any further commands passed by the XCMD, will be processed after the GoTo or GoSub line has been reached, and before any other script execution. This may cause unpredictable results if it is not catered for.

It is also not possible to reference any labels that are embedded within the library routines. Code within library routines is executed line by line normally, but can never be seen by the main script loop if back or forward referenced. The commands within the library script, and in fact any commands issued from any XCMD, are only transient in memory, and cannot reference labels or loops from within themselves.

Apart from these limitations you will find most script commands will perform as expected. A little testing will soon show what is possible and what is not. A useful tip is to always use the ‘$Quote’ substitution rather than hard coding a particular quote. This will make sure that your library routines work regardless of what the current Quote character may be. The same consideration of course applies to the Token character.

Multiple commands on the same script line are respected. If there are a series of commands separated by a ‘;’ they will be executed one by one in turn. If a conditional test fails within the line, further commands on that line will not be processed.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop October 1994
Updated by: Ewen Wannop October 2000

This technical note describes the Lister XCMD

The Lister XCMD

The Lister XCMD provides a method for scripts to maintain and work with lists. These are displayed in a standard dialog using a standard list box. Single and multiple choices may be made from the list, and the script can interrogate the results.

Up to 10 lists can be created at any one time.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT Lister 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Create List)

Function 1 creates a List:

```
EXT Lister 1  <ListNum> <Elements> {ElementSize} {MaxElements} {VarName}
EXT Lister 1 1 32 50 200 Result
```

The <ListNum> range is from 0-9. <Elements> range from 0-MaxElements which is a function of ElementSize. The optional {ElementSize} ranges from 1-244. It defaults to 50 if the optional size is not present. Note that a list can be created with no elements. In this case use Function with an Index of '0' to add multiple elements to the end of the empty list. Once elements have been added, the elements can then be changed using Function 3 using a specific Index value.

If the optional MaxElements is given, the list will be created to that size. If the list cannot be created to that size, MaxElements will be set as large as possible (default state). This is the maximum number of elements that can be contained within this list and is a function of ElementSize. Therefore MaxElements = 65535/(ElementSize+11). Note that to keep memory use as low as possible, you should specify ElementSize and MaxElements wherever possible.

If the optional {VarName} is given it will return the value for MaxElements.

The Failed flag is set if the list already exists, or cannot be created through insufficient memory.

Function = 2 (Clear List)

Function 2 clears a List:

```
EXT Lister 2  <ListNum>
EXT Lister 2 1
```

The <ListNum> Handle is cleared and all memory is disposed of. The Failed flag is set if the <ListNum> did not exist. If the <ListNum> was currently being displayed in a List window, the window will also be closed.

Function = 3 (Set Item)

Function 2 sets the values for an item in the list:

```
EXT Lister 3 <ListNum> <Index> “String” {Value} {Selected}
EXT Lister 3 1 1 “List Item” 0
```

If <ListNum> does not exist the Failed flag is set. If <Index> is greater than <Elements> the Failed flag is set. If <Index> = 0 the item is added to the end of the list increasing <Elements> by 1, but if the list is full (Elements = 256) the Failed flag is set and the item is not added to the list. If the "String" is longer than ElementSize it is truncated. The optional {Value} is a 32 bit number. The optional {Selected} can be 0 to leave unselected or 1 to select this item.

Function = 4 (Get Item)

Function 4 gets the values from an item in the list:

```
EXT Lister 4 <ListNum> <FlagLtr> <Index> <Var1forString> {Var2forValue}
        {Var3forSelected}
EXT Lister 4 1 1 Var1 Var2 Var3
```

If <ListNum> does not exist the Failed flag is set. The <FlagLtr> indicates how the Get Item command will select the item. If FlagLtr=N then the search will start at the next entry from the Index. If FlagLtr=I then the command will retrieve the entry indicated by <Index>. If <Index> is greater than <Elements> the Failed flag is set. The {Var1forString} is set to the string for this item. The optional {Var2forvalue} is set to the stored value of this item. The optional <Var3forSelected> is set to 1 if it the item is selected and 0 if not. If <FlagLtr>=I and <Index> = 0, then the first selected item in the list is returned and <Var3forSelected> is set to the <Index> value of that item.

Function = 5 (Delete Item)

Function 5 deletes an item from the list:

```
EXT Lister 5 <ListNum> <Index>
EXT Lister 5 1
```

If <ListNum> does not exist the Failed flag is set. If <Index> is greater than <Elements> the Failed flag is set. The item <Index> is removed reducing <Elements> by one.

Function = 6 (Sort List)

Function 6 sorts the list:

```
EXT Lister 6 <ListNum> <Field> {Direction} {Position}
EXT Lister 6 1 1
```

If <ListNum> does not exist the Failed flag is set. <Field> = 1 to sort on the <string> field and = 0 to sort on the <value> field. If the optional {Direction} is present the list will be sorted according to this value. The default Direction value = 0 and sorts from the top down. Direction = 1 sorts from the bottom up. The sort will start at the optional {Position} into the string entry. If this value is not given, or is = 0 the sort will start at the first character in the string. If <Field> = 0 this value will be ignored. If {Position} is greater than ElementSize the sort process will leave the list unchanged.

Function = 7 (Load List)

Function 7 loads a list from a space and CR delimited file:

```
EXT Lister 7 <ListNum> “FilePathName” {StartOffset}
EXT Lister 7 1 “:Volume:File”
```

If <ListNum> does not exist the Failed flag is set. If the file does not exist, or cannot be loaded, the Failed flag will be set. The file must consist of lines of data representing the data for the list. The items must not exceed the values for the target list. It should be constructed in the same way as the input to Function 3, spaces separating each entry and a CR separating each line.

```
<Index> “String” {Value} {Selected}<CR>
```

The items within the file are inserted into the target list at the Index position in the list. If this position is greater than the number of Elements in the list, it will be added to the end of the list increasing its size.

If the optional {StartOffset} is present it will be added to the Index number to determine where the entries will be entered into the list. The position will be {Offset+Index-1}.

Be careful that the numbering of the items is sequential. If a larger Index number preceeds a lower Index number, and the list has to be increased in size, you may find that the earlier entries are overwritten. The Save List function will always produce a file with a properly incremented list.

Function = 8 (Save List)

Function 8 saves a list to space and CR delimited file:

```
EXT Lister 8 <ListNum> “FilePathName” {StartValue}
EXT Lister 8 1 “:Volume:File” {1}
```

If <ListNum> does not exist the Failed flag is set. If the file exists the Failed flag will be set. The file will be written with lines representing the data from the list. The file will be constructed in the same way as the input to Function 7, and the data structure is the same as Function 3. Spaces separate each entry and CR’s separate each line.

The default Index number is 1, but if the optional {StartValue} is given, then numbering will start with that number. If the optional {StartValue} is 0, then all the entries will be numbered as 0. This will allow loading of the file at the end of an existing list.

Function = 9 (Show List)

Function 9 displays a list to the SHR screen in a dialog window:

```
EXT Lister 9 <ListNum> <Selection> “PromptString” “Button1Text” {“Button2Text”}
                {“Button3Text”} {“Button4Text”} {“Button5Text”} {“Button6Text”}
                {“Button7Text”} {“Button8Text”}
EXT Lister 9 1 1 “Please select an Item” “OK” {“Cancel”} {“Option”} etc.
```

If <ListNum> does not exist the Failed flag is set. <Selection> = 1 for single selections and = 2 for multiple selection. “PromptString” will be truncated if insufficient space is available in the window. The maximum size for the widest window is 60. ButtonTexts will be truncated if they exceed 9 characters. The $Hit substitution will be set to indicate the button that was hit. The range returned is 1-8. Button 1 will always be made the default button.

Key equivalents will be assigned where possible to the buttons from their PromptStrings. Button 1 however, being the default button, will always respond to <CR>. Button 2 will default to <Escape> if it cannot be assigned, and Button3 to <Space>.

If Button#Text is empty, denoted by a pair of QuoteCharacters only, the button will be invisible.

The SHR desktop will be selected if necessary to display the List dialog. The text screen will be restored when the dialog is removed.

Function = 10 (Set Window Size)

Function 10 sets the size and position of the dialog window:

EXT Lister 10 <ListNum> <Top> <Left> <ListSize> <ListWidth>
EXT Lister 10 1 22 166 12 50

The coordinates place the dialog within the desktop window. No checks, other than maximum and minimum limit checks, are made when this call is made. If the position results in the dialog being too small to contain the buttons and list control when the Show call is made, the Show call will fail.

Due to the Menu bar and the frame width of the dialog, the minimum top/left position is 18/8 pixels. The maximum sizes will depend on the ListSize and ListWidth. ListSize range is 4-12, and ListWidth range is 6-50. Setting a ListWidth smaller than the actual text length in the list may cause squeezing and truncation of the text. Setting a narrow window when using more than one button may cause some buttons not to display. They will still respond though to their key equivalents.

If this call is not made for a selected list, default sizes will be used placing the dialog centrally on the screen. It is only necessary to call Set Window Size if you wish to change the default position or size of the dialog. Any values set by this call will be lost when Clear List is called.

Function = 11 (Hide List)

Function 11 hides a list window:

EXT Lister 11 <ListNum>
EXT Lister 11 1

The target list window is closed if it was currently open. The failed flag is set if the <ListNum> did not exist.

Function = 12 (Select Item)

Function 12 selects and hilites the Indexed item within the target list:

EXT Lister 12 <ListNum> <Index>
EXT Lister 12 1 1

The Failed flag is set if the <ListNum> or the <Index> item do not exist.

Function = 13 (DeSelect Item)

Function 13 deselects and unhilites the Indexed item within the target list:

EXT Lister 13 <ListNum> <Index>
EXT Lister 13 1

The Failed flag is set if the <ListNum> or the <Index> item do not exist.

Function = 14 (Hilite Buttons)

Function 14 dims or undims buttons within the displayed list:

EXT Lister 14 <ListNum> <DimFlag> <Button#> {Button#} etc.
EXT Lister 14 1 1 1

Where:<DimFlag> = 0 shows normal active button
<DimFlag> = 1 inactivates button

Button numbers can be in any order in the command. If the List window is open when this command is called, the buttons will be changed immediately. A working copy for the target list is kept of the DimFlag settings as changed by this command. This template will then be used whenever a window is opened. The target list template is reset when a new List is Created or an old list is Cleared.

To reset all the buttons in a displayed list, the following command should be used:

EXT Lister 14 <ListNum> 0 1 2 3 4 5 6 7 8

The Failed flag is set if any parameter is out of range.

Function = 15 (Select Item)

Function 15 selects and scrolls to an item within the displayed list:

EXT Lister 15 <ListNum> <Index>
EXT Lister 15 1 1

If the current list is not being displayed, this call will do nothing. If the list is being displayed, then all other items will be deselected and the <Index> item will be selected and displayed.

The Failed flag is set if any parameter is out of range.

Function = 16 (Move Item)

Function 16 moves an item within a list:

EXT Lister 16 <ListNum> <Index> <Position>
EXT Lister 16 1 10 2

If <ListNum> does not exist the Failed flag is set. If <Index> is greater than <Elements> the Failed flag is set. The item <Index> is moved and inserted at the new position <Position>.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop May 1995
Updated by: Ewen Wannop May 8th 1995

This technical note describes the ResEdit XCMD

The ResEdit XCMD

The ResEdit XCMD allows editing and examination of the resource fork of a file.

Note: This is a potentially extremely dangerous XCMD, and if used without due care, can damage the resource fork of any file in your system. It should only be used with very great caution, and only if the script author is fully conversant with the structure of resources and the resource fork. It is advisable only to install or activate this XCMD if you are absolutely sure of the provenance of any scripts you might run. Never run a script that has come from an unknown source if this XCMD is present. Such a script might have all the effects of a dangerous virus. You have been warned!

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT ResEdit 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (LoadResource)

Function 1 loads the data from a specified resource from a file:

```
EXT ResEdit 1 ":PathName:FileName" ResourceType ResourceID ScriptEditor
EXT ResEdit 1 ":PathName:FileName" ResourceType "ResName" ScriptEditor
EXT ResEdit 1 ":Hard.Disk:TestFile" $$8006 $$00000001 ~E01234
```

Where: ResourceType = the target resource type
ResourceID = the ID of the target resource

Note that ResourceID can be given either as a number or a "ResName".

If the resource is found, the data block of the resource will be loaded and returned in the ScriptEditor.

If the file does not exist, or is already open, or does not have a resource fork, or the resource does not exist, or the EditorHandle does not exist or is invalid, or any input error occurs the Failed flag will be set.

Function = 2 (SaveResource)

Function 2 saves the data to a specified resource in a file:

```
EXT ResEdit 2 ":PathName:FileName" ResourceType ResourceID Attributes ScriptEditor
EXT ResEdit 2 ":PathName:FileName" ResourceType "ResName" Attributes ScriptEditor
EXT ResEdit 2 ":Hard.Disk:TestFile" $$8006 $$00000001 $0000 ~E01234
```

Where: ResourceType = the target resource type
ResourceID = the ID of the target resource
Attributes = resource attributes

Note that ResourceID can be given either as a number or a “ResName”. If a “ResName” is given a unique ID will be generated for the saved resource.

If the file exists or is already open, and has a resource fork, the contents of the ScriptEditor will be saved as a new resource. If a resource of the same type and number already exists, or if the “ResName” already exists, the resource will not be written and the failed flag will be set.

If the file does not exist, or does not have a resource fork, or the EditorHandle does not exist or is invalid, or any input error occurs the Failed flag will be set.

Function = 3 (DeleteResource)

Function 3 deletes a specified resource from a file:

```
EXT ResEdit 3 “:PathName:FileName” ResourceType ResourceID
EXT ResEdit 3 “:PathName:FileName” ResourceType “ResName”
EXT ResEdit 3 “:Hard.Disk:TestFile” $$8006 $$00000001
```

Use this command to clear an existing resource before using the SaveResource command.

Note that ResourceID can be given either as a number or a “ResName”.

If the file does not exist, or is already open, or does not have a resource fork, or the resource does not exist, or any input error occurs, the Failed flag will be set.

Function = 4 (CreateResourceFork)

Function 4 creates an empty resource fork in a file:

```
EXT ResEdit 4 “:PathName:FileName” {FileType} {AuxType}
EXT ResEdit 4 “:Hard.Disk:TestFile” {$$FF} {$$0000}
```

If the optional FileType and AuxType are not present, and the file does not already exist, the filetype will be set $06 and the AuxType to $0000. If the file already exists it is not necesary to provide FileType and AuxType.

The file will be created if it does not already exist.

If the file already has a resource fork, or any input error occurs, the Failed flag will be set.

Function = 5 (GetResourceList)

Function 5 gets a list of all the resources in a file:

```
EXT ResEdit 5 “:PathName:FileName” ScriptEditor
EXT ResEdit 5 “:Hard.Disk:TestFile” ~E01234
```

The returned information is in the form of CR delimited records with Hexadecimal values:

ResType	ResID	ResAttributes	ResSize	{ResName}
$0000	$00000000	$0000	$00000000	{NameIfExists}

Note that on a large application this function may take several seconds to execute.

If the file does not exist, or is already open, or does not have a resource fork, or the EditorHandle does not exist or is invalid, or any input error occurs the Failed flag will be set.

Function = 6 (ClearResourceFork)

Function 6 clears an existing resource fork in a file leaving an empty fork:

```
EXT ResEdit 6 ":PathName:FileName"
EXT ResEdit 6 ":Hard.Disk:TestFile"
```

This function will generate an error if there is not a correctly formatted GS/OS resource fork in the target file. It will not delete the resource fork of a Macintosh file.

If the file does not exist, or does not have a resource fork, or any input error occurs, the Failed flag will be set.

Function = 7 (GetResourceName)

Function 7 gets the name of a specified resource from a file:

```
EXT ResEdit 7 ":PathName:FileName" ResourceType ResourceID ResultVariable
EXT ResEdit 7 ":Hard.Disk:TestFile" $$8006 $$00000001 Result
```

If the file does not exist, or is already open, or does not have a resource fork, or the resource does not exist, or any input error occurs the Failed flag will be set.

Function = 8 (SetResourceName)

Function 8 sets the name of a specified resource:

```
EXT ResEdit 8 ":PathName:FileName" ResourceType ResourceID "NewName"
EXT ResEdit 8 ":PathName:FileName" ResourceType "ResName" "NewName"
EXT ResEdit 8 ":Hard.Disk:TestFile" $$8006 $$00000001 "Resource Name"
```

Setting an empty name deletes the name resource for the specified resource.

If the file does not exist, or is already open, or does not have a resource fork, or the resource does not exist, or any input error occurs the Failed flag will be set.

General Notes

This XCMD returns the raw data from a resource in a ScriptEditor Handle. It is entirely up to the script author to make sure that any data that is written back as a resource to the file represents valid resource data. Failure to make sure the data is valid could result in a trashed application. This obviously could have catastrophic effects on the system.

It is obviously permissable to use any method you like to process this data. The data contains full 8-bit data and the ScriptEditor will retain this, just make sure that you preserve it in any processing you may undertake. Use 'Make Char' to generate non-AlphaNumeric bytes when constructing resources from within a script.

Do not give any scripts using this XCMD to another user unless you are very sure that you have thoroughly tested and debugged them. You could be responsible for trashing irreplaceable files.

You may use either Decimal or Hexadecimal values where a value input is asked for. To indicate a HexaDecimal number put a '$' at the start of the number. Note that you will need to enter the '$' twice in order for it not be parsed as a Variable.

If you enter a name for ResName, then the name must match the resource name exactly.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop
Updated by: Ewen Wannop

January 1995
October 12th 2002

This technical note describes the ScriptEditor XCMD

The ScriptEditor II XCMD

The ScriptEditor II XCMD extends Spectrum's ability to work with TextEdit Records. It works with the ten ScriptEditors within Spectrum and allows powerful processing of these records such as searching for specified text, copying selected text and changing font styles.

In all the functions, if a parameter is out of range, or a required parameter is missing, the Failed flag will be set.

Where <TargetEditor> is indicated, this can be either a single number (0-9) representing a Spectrum ScriptEditor, or an actual TEHandle represented by a string, eg. '~E01234'.
Note the '~' marker must be used to indicate a TextEdit Handle.

If a destination ScriptEditor (0-9) does not already exist, it will be created by the XCMD.

Version 2.0 of ScriptEditor enhances the existing XCMD with four new functions.
Version 2.6 has further new functions and alternatively accepts 'SE' in the EXT calls.
Version 2.7 adds further new functions.
Version 3.0 adds a new send to port function
Version 3.4 improves the formatting of text in Function 20

It should be noted that with Function 6, string extraction may cause problems if an embedded return is encountered. The returned variable will terminate the data prematurely. The new Function 10 allows extraction of such strings. Function 6 has not been changed to keep compatibilty with existing scripts. Embedded Quotes might still cause problems though, and there is no easy solution to that problem. It is suggested that the 'bullet' character be used as the current Quote character in such cases.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT ScriptEditor 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Search for string)

Function 1 searches the target ScriptEditor for the specified string:

```
EXT ScriptEditor 1 <TargetEditor> <String> {StartPos} {Occurence} <VarName>
EXT ScriptEditor 1 0 "Is this string there?" 1 FoundPosition
```

<TargetEditor> indicates which ScriptEditor (0-9 or TEHandle) is to be searched.
<String> is the target string to find.
{StartPos} if the optional StartPos is given, the search will start from that position. If the StartPos is

not given, then the search will start at the beginning of the ScriptEditor.
{Occurence} if the optional 'nth' Occurence is given, then the search will start at the StartPos and find the 'nth' occurence of the string and return the position of the 'nth' occurence in VarName.
<VarName> variable will be set to the position of the string if it was found. If the string is not found, then the Failed flag will be set and the <VarName> variable will be left untouched.
If StartPos = 0, then the number of occurences of the string will be returned in VarName instead.

Function = 2 (Copy string)

Function 2 copies a selected string from one ScriptEditor to another:

```
EXT ScriptEditor 2  <TargetEditor> <StartPos> <EndPos> <DestinationEditor>
EXT ScriptEditor 2 0 1 10 1
```

<TargetEditor> indicates the source ScriptEditor (0-9 or TEHandle).
<StartPos> is the start position of the string.
<EndPos> is the end position of the string. If this is greater than the actual end position, the actual end position will be used.
<DestinationEditor> The selected string will be copied from the <TargetEditor> and replace the entire contents of the <DestinationEditor>.

Function = 3 (Clear string)

Function 3 clears a selected string from a target ScriptEditor :

```
EXT ScriptEditor 3  <TargetEditor> <StartPos> <EndPos>
EXT ScriptEditor 3 0 1 10
```

<TargetEditor> indicates the source ScriptEditor (0-9 or TEHandle).
<StartPos> is the start position of the string.
<EndPos> is the end position of the string. If this is greater than the actual end position, the actual end position will be used.
The specified string will be removed and the TERecord will decrease in size accordingly.

Function = 4 (Paste string)

Function 4 pastes the contents of one ScriptEditor into another:

```
EXT ScriptEditor 4  <TargetEditor> <SourceEditor> <Position>
EXT ScriptEditor 4 0 1 10
```

<TargetEditor> indicates the ScriptEditor that will be pasted into.
<SourceEditor> the entire TERecord of the SourceEditor will be pasted into the TargetEditor at the indictated <Position>.
<Position> the point in the <TargetEditor> where the <SourceEditor> will be pasted. If this is beyond the end of the TERecord, it will be added to the end.

Function = 5 (Insert string)

Function 5 inserts an actual string into a ScriptEditor:

```
EXT ScriptEditor 5  <TargetEditor> "String" <Position>
EXT ScriptEditor 5 0 "This string will be inserted" 10
```

<TargetEditor> indicates the ScriptEditor that will be inserted into.
"String" is an actual string delimted by the current Quote Character.

<Position> the point in the <TargetEditor> where the string will be inserted. If this is beyond the end of the TERecord, it will be inserted at the end.

Function = 6 (Extract_string)

Function 6 extracts a selected string from a ScriptEditor to a Variable:

EXT ScriptEditor 6 <TargetEditor> <StartPos> <EndPos> <VarName>
EXT ScriptEditor 6 0 1 10 ResultVariable

<TargetEditor> indicates the source ScriptEditor (0-9 or TEHandle).
<StartPos> is the start position of the string.
<EndPos> is the end position of the string. If this is greater than the actual end position, the actual end position will be used.
<VarName> is the variable which the selected string will be extracted to.

Function = 7 (Set_string)

Function 7 places a selected string into a ScriptEditor:

EXT ScriptEditor 7 <TargetEditor> “String”
EXT ScriptEditor 7 0 “This string will replace contents”

<TargetEditor> indicates the source ScriptEditor (0-9 or TEHandle).
“String” is the string which will replace the entire contents of the <TargetEditor>.

Function = 8 (Set Font)

Function 8 changes the Font and Style of a selected string within a ScriptEditor:

EXT ScriptEditor 8 <TargetEditor> <StartPos> <EndPos> <Font> <Size> <Style> {ForeColor} {BackColor}
EXT ScriptEditor 8 0 1 10 “Times” 8 0 {0} {15}

<TargetEditor> indicates the source ScriptEditor (0-9 or TEHandle).
<StartPos> is the start position of the string.
<EndPos> is the end position of the string. If this is greater than the actual end position, the actual end position will be used.
<Font> indicates the Font that will be used. The Font can be defined by its name or the Family number given in decimal or hexadecimal format. use a ‘$’ to indicate a hexadecimal number. The Font need not exist in your system, but be aware that the TERecord will not display correctly if it is not present.
<Size> in points of the Font.
<Style> Bit 0 = Bold, Bit 1 = Italic, Bit 2 = Underline, Bit 3 = Outline, Bit 4 = Shadow
{ForeColor} is the optional color to use for the foreground color of the text. Black will be used if this is not specified.
{BackColor} is the optional color to use for the background color of the text. White will be used if this is not specified

Function = 9 (Get Line)

Function 9 gets a line from a ScriptEditor and places it in a variable:

```
EXT ScriptEditor 9  <TargetEditor> <StartPos> <VarName1> <VarName2>
EXT ScriptEditor 9 0 10 ReturnLine NextPosition
```

<TargetEditor> indicates the source ScriptEditor (0-9 or TEHandle).
<StartPos> is the start position of the string.
<VarName1> returns the line that was found. The line is either the string up to the next Carriage Return or the end of the TERecord.
<VarName2> is the position of the next character after the end of the returned string. By using this value as the <StartPos> input and calling this function once again, repeated calls may be made to return all the separate lines within a TERecord.

Function = 10 (ExtractString2)

Function 10 extracts a selected string from a ScriptEditor to a Variable:

```
EXT ScriptEditor 10  <TargetEditor> <StartPos> <EndPos> <VarName>
EXT ScriptEditor 10 0 1 10 ResultVariable
```

This Function performs in the same way as Function 6, but any characters below ASCII 32 are represented as their '^' equivalent in the output variable. It should be noted that the resulting variable may be larger then the difference between EndPos-StartPos if any characters have been represented as '^' equivalents.

Function = 11 (ConvertQuoted)

Function 11 converts a ScriptEditor to a quoted reply:

```
EXT ScriptEditor 11  <TargetEditor> {QuoteString}
EXT ScriptEditor 11 0 {> }
```

The TargetEditor will be split into quoted lines with a final maximum length of 'SendLines'. The default will be to add a default quote of '> ' to the start of each line, but if the optional {QuoteString} is given, this will be used instead. If a QuoteString of '^@' (ASCII 0) is given, then the text will be formatted with no added Quote string.

The Spectrum variable 'SendLines' is used to calculate the final quoted length. This will be the QuoteString plus the extracted characters. A CR will be inserted at the split point, and the split will be made at the nearest SPC to the SendLines length. The maximum value of 'SendLines' for this Function is limited to 636 characters. If it is greater than that value, the Failed flag will be set.

Note that as the normal use of this command will be to prepare reply text for transmission within an eMail message, the Function will remove any styles from the input record.

Function = 12 (AppendToFile)

Function 12 appends the text of a ScriptEditor to a file:

```
EXT ScriptEditor 12  <TargetEditor> <PathName>
EXT ScriptEditor 12 0 ":Hard.Disk:Folder:FileName"
```

Only TEXT files can be appended to, and as a result, any styles in the ScriptEditor will be lost.

Function = 13 (InsertFromFile)

Function 13 inserts a file into a target ScriptEditor :

```
EXT ScriptEditor 13  <TargetEditor> <PathName> <Position> {StartPosition} {Length}
EXT ScriptEditor 13 0 ":Hard.Disk:Folder:FileName" 10 {10} {2000}
```

Note that both TEXT and Teach files can be inserted. If a text file is inserted, then a default style of Shaston 8 will be used.

If the file is a TEXT file, and the optional {StartPosition} and {Length} are given, then only that part of the file represented by these values will be inserted. If the file is a TEACH file, these optional values will be ignored. The default without the optional values is to insert the entire file. Note that these optional values must be valid. If the StartPosition, or the StartPosition plus Length, is beyond the end of file, the Failed flag will be set.

Function = 14 (CopyToClipboard)

Function 14 copies selected text to the System Clipboard:

```
EXT ScriptEditor 14  <TargetEditor> <StartPos> <EndPos>
EXT ScriptEditor 14 0 1 10
```

This will replace any current contents of the Clipboard with the selected text. If this is greater than the actual end position, the actual end position will be used.

Function = 15 (SearchFile)

Function 15 searches a file for a target string:

```
EXT ScriptEditor 15  <PathName> <Direction> <TargetString> <StartPos> {Occurrence }
                                                                    <VarName>
EXT ScriptEditor 15 ":Hard.Disk:Folder:FileName" 0 "String" 1 1 FoundPosition
```

Where:Direction = 0 search forwards from the StartPos
Direction = 1 search backwards from the StartPos
TargetString = Max 256 characters
StartPos = 1 for the beginning of the file
Occurrence = 1 for the first occurrence etc. (Range 1-65535)

StartPos will be set to the end of the file before starting the search, if that is less than the given position. The optional {Occurrence} value will be set to 1 if not specified. The returned VarName will hold the file position at the next character after where the target string was found. If the string is not found, then VarName will be set to '0'. The current state of the CaseSense flag will be respected.

NOTE: The Failed flag will not be set if the string is not found, you must check the contents of VarName for this.

Although strictly speaking this is not a Function that operates on a ScriptEditor, I had nowhere else to put it easily! This enhanced File Search Function will probably be included in Spectrum v2.2 and later.

During the search, the file will be opened as Read only. This allows this Function to be used on an already open file. The file will be closed at the end of the search process.

If the file is not found, cannot be opened for reading, or any other error occurs, then the Failed flag will be set.

Function = 16 (ConvertToHex)

Function 16 converts a target range of bytes into their Hex equivalent:

```
EXT ScriptEditor 16  <TargetEditor> <StartPos> <Range> <VarName>
EXT ScriptEditor 16 0 1 12 Result
```

As a Hex representation of a single character or byte will take two bytes, the resulting string will be twice the size of <Range>. This limits <Range> to 1-128.

If any error occurs, then the Failed flag will be set.

Function = 17 (ConvertFromHex)

Function 17 converts a target string of Hex values into the contents of an EditorHandle:

```
EXT ScriptEditor 17  <TargetEditor> “String”
EXT ScriptEditor 17 0 "00112233445566778899AABBCCDDEEFF"
```

The contents of the ScriptEditor will be replaced by the converted values from the supplied string. The first or lowest byte is at the start of the string. All bytes are represented by a pair of ASCII characters.

If the string is not an even number of bytes, or any other error occurs, then the Failed flag will be set.

Function = 18 (ReverseData)

Function 18 reverses the order of bytes or characters in the contents of an EditorHandle:

```
EXT ScriptEditor 18  <TargetEditor>
EXT ScriptEditor 18 0
```

If the string is greater than 65535 bytes, or any other error occurs, then the Failed flag will be set.

Function = 19 (SearchFile2)

Function 19 searches a file for a target string:

```
EXT ScriptEditor 19  <RefNum> <Direction> <TargetString> <StartPos> {Occurrence }
                                                                    <VarName>
EXT ScriptEditor 19 1 0 "String" 1 1 FoundPosition
```

This Function is the same as Function 15, expect that it uses the RefNum of an open file instead of a PathName. The RefNum of an open file can be obtained using the $FileID# replacement item.

If the file is not open, or any other error occurs, then the Failed flag will be set.

Function = 20 (ExtractMessage)

Function 20 searches a file for a target string and extracts data:

```
EXT ScriptEditor 20  <RefNum> <TargetString> <StartPos> <Format> <TargetEditor>
                                <VarName> {VarName1} {VarName2} {VarName3}
EXT ScriptEditor 20 1 "String" 1 1 0 FoundPosition {Line1} {Line2} {Line3}
```

Where:Format = 1 no formatting
Format = 2 text flowed
Format = 3 no formatting (VarStrings included)
Format = 4 text flowed <VarStrings included)

This Function is similar to Function19, but is designed for use with Offline Message Readers. It will search from the <StartPos> to the <TargetString>, or end of file, and extract the data between these two points in the following way:

All leading spaces and CRs will be stripped to the actual text characters before extraction. Any controls, other than CRs or Tabs, in the entire text will also be stripped. If any of the three optional {VarNameX} exists, a line of text will be extracted for each variable. The remaining text will be put into the <TargetEditor>. Optionally the contents of these variables can be included within the <TargetEditor> but only any trailing text will be formatted if the <Format> flag has been set. The formatted text will be flowed into paragraphs. If this text exceeds 64K, then it will not be formatted.

If there is insufficent memory for the message to be extracted, the TargetEditor will be cleared. If <VarName1> has been defined, this variable will return with the message "^E*** Memory Error ***".

If the file is not open, then end of file has already been reached, or any other error occurs, then the Failed flag will be set.

Function = 21 (CopySelection)

Function 21 copies selected text from one Editor to another:

```
EXT ScriptEditor 21  <SourceEditor> <TargetEditor> {Value}
EXT ScriptEditor 21 ~E01234 0
```

Where:No Value = selected text
Value = 1 entire record
Value = 2 selected text

To specifiy a specific TextEdit record, the actual Handle must be given for the <SourceEditor> in the format ~E01234. If however a value of 0 is given, then the current active TextEdit will be used. The main purpose of this Function is to easily extract selected text from active TextEdit controls.

If an error occurs, then the Failed flag will be set.

Function = 22 (SetSelection)

Function 22 sets the selection points in a TextEdit record:

```
EXT ScriptEditor 22 <TargetEditor> <StartPos> <EndPos>
EXT ScriptEditor 22 0 1 10
```

This Function allows a selection to be made in active TextEdit controls.

If an error occurs, then the Failed flag will be set.

Function = 23 (GetSelection)

Function 23 gets the selection points in a TextEdit record:

```
EXT ScriptEditor 23 <TargetEditor> <VarName> <VarName1>
EXT ScriptEditor 23 0 Start End
```

This Function returns the current selection points of a TextEdit record.

If an error occurs, then the Failed flag will be set.

Function = 24 (SendScriptEditor)

Function 24 sends a TEHandle directly to the port:

EXT SE 24 <TEHandle>
EXT SE 24 0
EXT SE 24 ~E01234

The supplied TEHandle will be sent directly to the port in a constant stream without being processed in the usual way by Spectrum. The only Spectrum settings that will be relevant are the Xon/Xoff flag and the baud rate setting. This differs from the 'Send ScriptEditor' command which is passed through various filters and other settings within Spectrum.

If the TEHandle is not a valid handle, or any error occurs, the Failed flag will be set.

Function = 25 (DisplayHTML)

Function 25 displays a TEHandle in the HTML.Engine display:

EXT SE 25 <TEHandle>
EXT SE 25 0
EXT SE 25 ~E01234

The supplied TEHandle will be sent to the HTML.Engine if it is present in the 'Add.Ons:Drivers' folder. This display shuts down the MenuBar and displays any enclosed <HTML> text. Click the Close box to shut down the window and restore the MenuBar.

A script should verify the presence of the bounding <HTML> and </HTML> tags before passing the Handle.

If the TEHandle is not a valid handle, or any error occurs, the Failed flag will be set.

Function = 26 (EncodeBase64)

Function 26 encodes the supplied ScriptEditor into Base64 format:

EXT SE 26 <SourceTEHandle> <TargetTEHandle> {BreakChar(s)}
EXT SE 26 ~E01234 ~E04321 {13}

The encoded text will be returned as one long block of Base64 characters. If the optional {BreakChar(s)} is given, then the encoded string will be broken after every 76 characters with the BreakChar(s).

The long decimal value for BreakChar(s) will be converted to Hexadecimal internally, so to give two BreakChars, such as CR/LF, you supply a value in excess of 256 with the low byte being inserted first. The decimal equivalent of CR/LF ($0A0D) is therefore 2573. You can use HodgePodge functions of you need to convert Hexadecimal numbers to decimal.

The Failed flag will be set if any error occurs.

Function = 27 (DecodeBase64)

Function 27 decodes the supplied ScriptEditor from Base64 format:

EXT SE 27 <SourceTEHandle> <TargetTEHandle>
EXT SE 27 ~E01234 ~E04321

The Failed flag will be set if any error occurs.

Function = 28 (InsertFromFile)

Function 28 inserts from a file into a target ScriptEditor :

```
EXT ScriptEditor 28  <TargetEditor> <FileID> <Position> {StartPosition} {Length}
EXT ScriptEditor 28 0 1 10 {10} {2000}
```

This is similar to Function 13, but will only insert from the data fork of an open file.

If the optional {StartPosition} and {Length} are given, then only that part of the file represented by these values will be inserted. The default without the optional values is to insert the entire file. Note that these optional values must be valid. If the StartPosition, or the StartPosition plus Length, is beyond the end of file, the Failed flag will be set.

Function = 29 (ConvertQuoted2)

Function 29 converts a ScriptEditor to a quoted reply:

```
EXT ScriptEditor 29  <TargetEditor> {QuoteString}
EXT ScriptEditor 29 0 {> }
```

This Function works exactly like Function 11, except that where a line exceeds the value of SendLines, and so is split at the previous space, the space will be preserved, and the CR will be inserted after the space. In Function 11, the space was simply replaced by the CR.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop January 1995
Updated by: Ewen Wannop January 1995

This technical note describes the Speech XCMD

The Speech XCMD

The Speech XCMD gives Spectrum the ability to speak. It uses the Speech Tools available from ByteWorks. These tools are not provide with Spectrum and must be obtained separately from an appropriate supplier. Without these tools being present in the system when Spectrum is started, the Speech XCMD will deactivate itself, and be removed from memory.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT Speech 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Set Speech Settings)

Function 1 sets the settings used when speaking text:

```
EXT Speech 1 {Gender} {Tone} {Pitch} {Speed} {Volume} {MaxVolume}
EXT Speech 1 0 0 5 5 5 1
```

{Gender} 0 = Male, 1 = Female, default to 0
{Tone} 0 = Bass, 1 = Treble, default to 0
{Pitch} range 0-9, default to 5
(Speed} range 0-9, default to 5
{Volume} range 0-9, default to 5
{MaxVolume} 0 = use system volume, 1 = use max system volume, defaults to 1

MaxVolume is provided to make sure that the speech will be heard on a system that may have a low system volume set. The actual volume you will hear the speech is then controlled by the Volume parameter within the XCMD.

Note that these settings are not saved to disk and so are temporary in memory only. This allows you to save the default settings using Function 5, temporarily change some settings with this command, and then restore your default settings with Function 4.

The Failed flag will be set if any one of the optional parameters is out of range.

Function = 2 (Speak Text)

Function 2 speaks a give string of text:

```
EXT Speech 2 “String”
EXT Speech 2 “What does this sound like!”
```

Speaks the text string. Text will be spoken according to its capitalisation and punctutation. You should refer to the instructions for the Speech Tools for further information.

Function = 3 (ShutDown Tools)

Function 3 shuts down the Speech Tools:

```
EXT Speech 3
EXT Speech 3
```

The Speech Tools are started the first time that Function 2 is called. They are shut down automatically when a script stops. They can be shut down at any time by calling this function. They will be restarted if ncessary the next time you call Function 2. You may wish to shut the tools down as soon as you have spoken your text as they can interfere with the correct operation of the Sound Tools.

Function = 4 (Load Settings)

Function 4 loads the settings from a disk file:

```
EXT Speech 4
EXT Speech 4
```

If the file does not exist the failed flag will be set.

Function = 5 (Save Settings)

Function 5 saves the settings to a disk file:

```
EXT Speech 5
EXT Speech 5
```

If this file exists when Spectrum is next started and the Speech XCMD loads, these settings will be used as the new default settings. The Speech.Prefs file can be found in the Add.Ons:XCMDs folder. You can delete this file to restore the original default settings.

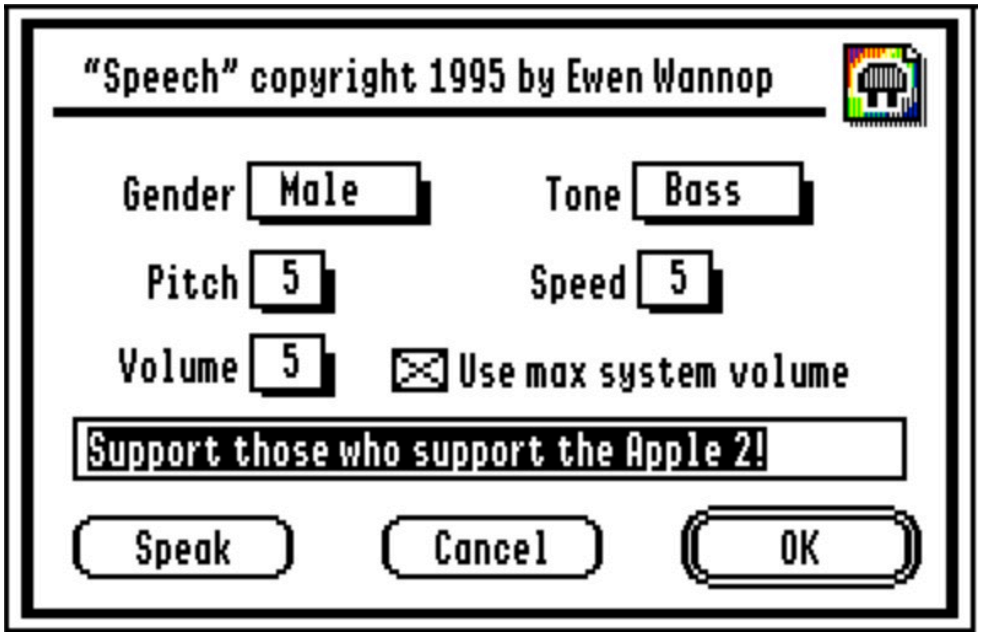
The Failed flag will be set if it was not possible to save the settings to disk.

Manual Settings Control

The Speech settings can be manually controlled from a dialog window. This window is called up from a script using the following command:

Ext Speech

Note that no function number is given for this command. If the SHR 640 desktop is not currently showing, this command will do nothing.



The Speech settings window is mainly self explanatory. You can edit a sample line of text and hear how the various settings will affect the sound output. Simply click the 'Speak' button to hear the text. When you click the 'OK' button, the settings will be saved to disk. These settings will then be used as the default settings the next time that Spectrum starts and loads the Speech XCMD. If you click the 'Cancel' button, the settings will not be saved.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop February 5th 1998

This technical note describes the TopCat XCMD

The TopCat XCMD

The TopCat XCMD allows Spectrum v2.1 to connect to an Internet Service Provider through the 'Marinetti' TCP/IP CDev written by Richard Bennett. This CDev is available from Delphi, Genie and Richard's homepage. For full functionality, you will need to register the CDev directly with Richard.

The functionality of the TopCat XCMD will be included within any future version of Spectrum, so the XCMD is specific to Spectrum v2.1 only.

Using suitable scripts and supporting XCMDs and XDisplays, TopCat will allow full Internet access from Spectrum v2.1. For instance, a Telnet client can be written in one script line...

If the 'Marinetti' CDev is not installed, the TopCat XCMD will not load. If the Spectrum port has been set to a slotted Serial Card such as the Super Serial Card, all Functions from Function 1 to 11 will fail. If Function 1 is not first successfully called, all Functions from 3 to 11 will fail.

Version 2.0 (or later) of TopCat is required for the Toolbox version of Marinetti (v1.2 or later).

NOTE: It is important for the proper operation of Spectrum, and for the non-destructive switching of the serial ports, that you do not start up TCP/IP from the Marinetti CDev before you run Spectrum. It is also important that you use care if any other TCP/IP application are run while Spectrum is active. Spectrum turns off its normal use of the serial port, only while the TopCat XCMD is in control. A port conflict amy occur if any other application, or the CDev itself, turns on TCP/IP while Spectrum has control of the ports.

They said it could never be done. Then the A2 gods created SIS, and Richard worked a miracle...

Function = Null (Manual Control Dialog)

With no function specified, and with the SHR 640 desktop showing, a dialog window will open allowing you to manually Telnet to a host. If the host is successfully connected to, the online display will be opened. It is suggested you select either the Text or Spectrum SHR display, so you will be able to use the Chatline without turning on half duplex control.

```
EXT TeeCee
```

If a connection was successful, the address will be saved in the file ‘TopCat.Prefs’ in the Add.Ons folder. If this file exists when TopCat is next loaded, the saved value will be loaded again. Delete the file ‘TopCat.Prefs’ if you wish to remove the Telnet host you last used. A default address for Delphi will be presented if you enter a bad address.

To close the connection, call up this Function once again to select the Disconnect button.

The dialog is also available from the Special Black Apple Menu as 'TCP/IP Telnet...'.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT TeeCee 0 {VarName}
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Connect)

Function 1 connects the XCMD to the TCP/IP CDev:

```
EXT TeeCee 1
EXT TeeCee 1
```

You must have correctly installed the Marinetti CDev with your ISP number and connection script before you can call any of the TopCat functions. Function 1 must be called before any other Functions. This Function redirects all port vectors within Spectrum into the XCMD thus freeing the serial port for Marinetti to use. Until Function 3 has been called, no normal input or output will be possible from Spectrum. No incoming data will be seen, and all outgoing data will be ignored.

If the TCP/IP CDev is not already connected, a full dial/connect sequence will be called by Marinetti.

The Failed flag will be set if any error occurs.

Function = 2 (DisConnect)

Function 2 disconnects the XCMD from the TCP/IP CDev:

```
EXT TeeCee 2
EXT TeeCee 2
```

This function closes all current sockets, disconnects the TCP/IP Cdev from the ISP if it was active, restores the port settings within Spectrum, and then returns normal functionality to Spectrum. Spectrum has control of the serial port once more.

If TopCat caused the TCP/IP CDev to dial and connect, and no other application is currently using it, then a full disconnect will be made. Function 1 must be called again before any other TopCat Functions will be active.

Note: If TCP/IP cannot be disconnected, the failed flag will be set, and the normal functionality of Spectrum will not be restored.

The Failed flag will be set if any error occurs.

Function = 3 (OpenSocket)

Function 3 opens a socket with the target host:

```
EXT TeeCee 3 <HostIP,Port> <Mode> <VarName>
EXT TeeCee 3 "194.83.84.1,23" 1 SocketNumber
```

Where:HostIP given in dotted decimal with Port spaced with comma.
(use "199.93.4.65,23" for Delphi)

Port = 23 for Telnet (Default Port if one is not supplied)
Port = 80 for the Web

Mode = 0 data returned through normal port vectors
Mode = 1 data returned through normal port vectors (Telnet mode)
Mode = 2 data only returned/sent by Functions 7 through 10

The returned VarName will hold the current socket number. Up to 16 active sockets can be currently handled by TopCat. This socket will then be made the currently active socket.

NOTE: If the Host breaks the connection, the active socket may be cancelled. This will result in the Failed flag being set if you then call Function 4 or 5 for that SocketNumber. You should monitor the Failed flag after each of these calls to make sure that you are still connected. At any other time, you may call Function 6 to check if the socket is still connected.

The TCP/IP CDev must be connected before this Function, so you must call Function 1 before you can open a socket. Once a socket has been opened, all input and output from Spectrum will be directed through the active socket. This will be either through the normal port vectors or through Functions 7 and 8.

The port vectors are simply the entry points that Spectrum normally uses. It means that any incoming data will be directed to the screen, outgoing data will go to Marinetti, and script commands will work normally.

The Failed flag will be set if any of the required parameters are invalid, are not present, or any other error occurs.

Function = 4 (CloseASocket)

Function 4 closes an opened socket:

EXT TeeCee 4 <SocketNumber>
EXT TeeCee 4 32

If SocketNumber = 0 then all sockets will be closed.

The Failed flag will be set if any of the required parameters are invalid, are not present, or any other error occurs.

Function = 5 (ChangeActiveSocket)

Function 5 changes the active socket:

EXT TeeCee 5 <SocketNumber> {Mode}
EXT TeeCee 5 33 {1}

All input and output from Spectrum will be directed through the new active socket. The optional {Mode} value can be applied to change the current operating mode of a socket.

The Failed flag will be set if the socket has not been opened, any of the required parameters are invalid, are not present, or any other error occurs.

Function = 6 (SocketStatus)

Function 6 checks whether a Socket is still connected or not:

```
EXT TeeCee 6  <SocketNumber> {TCPState}
EXT TeeCee 6 33 {Result}
```

If the optional {TCPState} variable is not given, then the Failed flag will be set if the socket is closed. Otherwise it will return the current TCP state in the variable. It is then up to the controlling script to handle events from there on.

The Failed flag will be set if the Socket is not a valid SocketNumber.

Function = 7 (GetHandle)

Function 7 fills the supplied ScriptEditor handle with data:

```
EXT TeeCee 7 "LineEndMarkerString" <EditorHandle>
EXT TeeCee 7 "^M" ~E01234
```

Normally data is returned in the incoming port flow. If Mode 2 has been selected for the currently active session, no data will be returned through the port. You must call this Function or Function 9 to retrieve data.

All data up to, but not including the LineEndMarkerString, will be returned. If the LineEndMarkerString is not seen, no data will be returned, and any partial line data will be stored until the next time that Function 7 is called. If the LineEndMarkerString is empty, then all waiting data will be returned. The maximum length of the LineEndMarkerString is 32 characters. Any existing contents of the EditorHandle will be lost.

It should be noted that this Function is active in all modes, and will return any data that has not yet been retrieved from Marinetti.

If no complete line can be returned, or the EditorHandle is not valid, the Failed flag will be set.

Function = 8 (SendHandle)

Function 7 sends the data from the supplied ScriptEditor handle:

```
EXT TeeCee 8 <EditorHandle>
EXT TeeCee 8 ~E01234
```

The data contents of the supplied ScriptEditor handle will be sent as a block. Any LineEndMarkers must be supplied within the Handle.

It should be noted that this Function is active in all modes, and will send data immediately to Marinetti.

If the EditorHandle is not valid, or is empty, the Failed flag will be set.

Function = 9 (GetVar)

Function 9 gets a line of data into the supplied VarName:

```
EXT TeeCee 9 "LineEndMarkerString" "VarName"
EXT TeeCee 9 "^M" Result
```

Normally data is returned in the incoming port flow. If Mode 2 has been selected for the currently active session, no data will be returned through the port. You must call this Function or Function 7 to retrieve data.

All data up to, but not including the LineEndMarkerString, will be returned. If the LineEndMarkerString is not seen, no data will be returned, and any partial line data will be stored until the next time that Function 9 is called. If the LineEndMarkerString is empty, then all waiting data up to a maximum of 256 characters will be returned. The maximum length of the LineEndMarkerString is 32 characters, and the returned String will not exceed 256 characters. If you think the returned data might exceed that length, before the LineEndMarkerString is seen, use Function 7, or you will not be able to retrieve any data at all without using an empty LineEndMarkerString.

The contents of VarName will be reset. If no line can be returned because the marker was not seen, but there is data in the buffer, the VarName will be returned as an empty String. You can retrieve the data by calling the Function again, with no LineEndMarkerString, or keep trying till more data has arrived.

It should be noted that this Function is active in all modes, and will return any data that has not yet been retrieved from Marinetti.

If there is no waiting data, or any other occurs, then the Failed flag will set.

Function = 10 (SendString)

Function 10 sends the data from the supplied String:

```
EXT TeeCee 10 "String"
EXT TeeCee 10 "This line will be sent"
```

The supplied String will be sent as a block. Any LineEndMarkers must be supplied within the String. The maxiumum length of String allowed is 256 characters.

It should be noted that this Function is active in all modes, and will send data immediately to Marinetti.

If any error occurs, the Failed flag will be set.

Function = 11 (FlushOutputBuffer)

Function 11 flushed any waiting data:

```
EXT TeeCee 11
```

Data output through the normal Spectrum vectors is stored in a 512 byte buffer, and is sent if this buffer becomes full, or at other times depending on the current 'Mode' or 'state'. This Function simply sends any waiting data immediately.

With Mode 0, data is not sent till the buffer fills, or Function 11 is called. With Telnet Mode 1, data is sent whenever a CR is seen in the output stream. With Mode 2, data is sent immediately the related Function is called.

If a protocol transfer is started by Spectrum, data is sent whenever Spectrum looks for input data. This is usually between packets, so the handshake works as expected. However transfers may not proceed as smoothly under TCP/IP as under normal port use. You may need to experiment to find the best method.

The Failed flag will be set if TopCat is curently offline.

Function = 12 (SetFlushCharacter)

Function 12 sets the character that is used to flush the output buffer:

```
EXT TeeCee 12 <Value>  Range 0 - 256
EXT TeeCee 12 13
EXT TeeCee 12
```

For Mode 1, data is normally flushed on a CR. For Mode 0, data is only flushed if the buffer fills, or Function 11 is called. If a value is set by this Function, then it will be used as the 'Flush' character for both Mode 0 and 1. If a value of 0, or no value is given, the default conditions will be reset.

The setting is an overall value and will apply to all sockets that are open.

The Failed flag will be set if any error occurs.

Function = 13 (SetEOLTranslation)

Function 13 sets the EOL translation:

```
EXT TeeCee 13 <Value1> {Value2}  Range 0 - 256 for each value
EXT TeeCee 13 13 {10}
EXT TeeCee 13 10
EXT TeeCee 13
```

For Modes 0 and 1, any CRs (ASCII 13) in the output stream will be converted into the value given here. This can be used to translate the normal CR that Spectrum outputs into an LF or CR/LF pair that many hosts require. If the value is the same as that given in Function 12, then a flush of the outbuffer will then be triggered. If a value of 0, or no value is given, for the first Value, then the default conditions will be reset. If a value of 0, or no value is given, for the second Value, then it will not be applied.

The setting is an overall value and will apply to all sockets that are open. It should be noted that the first of the Values will be the one used for subsequent checks for the Flush Character.

The Failed flag will be set if any error occurs.

Function = 14 (FlushFrequency)

Function 14 controls whether data is flushed character by character or by the EOL markers:

```
EXT TeeCee 14 <Value>  Range 1 to 512
EXT TeeCee 14 1
EXT TeeCee 14
```

For Modes 0 and 1, data is normally flushed on a flush trigger character, by Function 11, or when the 512K output buffer fills. If a Value is given, then all data will be sent when the buffer reaches the Value size. A Value of 1 will send character by character. This is a slower method than sending by the usual default methods, but might be suitable for certain situations. If a value of 0, or no value is given, the default conditions will be reset.

It should be noted that for Telnet mode 1, when a single character is normally being flushed, this will always result in the character plus the Telnet NOP ($FF + $F1) being sent as well. Some hosts, such as Delphi, do not respond to a single character. If the FlushFrequency has been set to 1, this process will be overidden, and a single character only will be sent.

Tip: By controlling the flush frequency, you may be able to fine tune an operation. Function 11 can be used at any time to flush the rest of the buffer whatever size it has been set to.

The setting is an overall value and will apply to all sockets that are open.

The Failed flag will be set if any error occurs.

Function = 15 (SetPrefs)

Function 15 sets the connect method that will be used by TopCat, and the default dotted address used for the manual control dialog:

```
EXT TeeCee 15 <Method> <HostIP,Port>
EXT TeeCee 15 4 "194.83.84.1,23"
```

Where: Mode = 1 Ethernet
Mode = 2 Local
Mode = 3 PPP
Mode = 4 SLIP

Note that Modes 1-3 are not yet activated within Marinetti at the time of writing.

The Failed flag will be set if any error occurs.

Function = 16 (SetClosedResponse)

Function 16 tells TopCat what to do if a socket closes:

```
EXT TeeCee 16 <SocketNumber> <"String">
EXT TeeCee 16 33 "This will be put in input stream"
EXT TeeCee 16 33 ""
```

If the SocketNumber is active, and subsequently closes, the "String" will be placed in the Input stream where a script can monitor for it. The script should then take appropriate action to close the port itself using Function 4. Entering an empty string cancels the action.

The port will be checked, and the possible response given, whenever Spectrum sends its 'Spectrum Is Idle' ICP call.

If the socket is not active, the Failed flag will be set.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop January 1995
Updated by: Ewen Wannop February 1995

This technical note describes the Twilight II XCMD

The Twilight II XCMD

The Twilight II XCMD works in conjunction with the Twilight II screen saver to inhibit screen blanks that could cause problems with Spectrum's downloading processes. It does not have any user interface other than to return its version number. It's sole purpose is to work in the background to control Twilight II so that it will not interfere with interrupt sensitive processes within Spectrum.

It does this by interpreting the various IPC calls sent by Spectrum, and sends appropriate responses directly to Twilight II as needed.

If you do not have Twilight II active on your system, this XCMD will do nothing. In which case you can safely remove it from the Add.Ons:XCMDs folder.

I the TW2.Prefs file exists when the XCMD is started, Twilight II will be inactivated while Spectrum is running. Normal settings will be restored when Spectrum or the XCMD is shut down.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT TwilightII 0 <VarName>
EXT TwilightII 0 VersionNumber
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (Set preferences)

Function 1 sets the state of the completely inactivate flag:

```
EXT TwilightII 1 <Inactive> <Blanking>
EXT TwilightII 1 1 1
```

Inactive = 1 inactivates Twilight II while using Spectrum.
Inactive = 0 makes Twilight II active as follows:
While online or when running a script:
Blanking = 0 uses background blanking
Blanking = 1 Inactivates Twilight II

The setting is not saved to disk and remains in memory only.

Function = 2 (Load preferences)

Function 2 load the saved preferences from disk and :

EXT TwilightII 2
EXT TwilightII 2

If the file TWII.Prefs file does not exist this command will do nothing.

Function = 3 (Save preferences)

Function 3 saves the preferences to disk:

EXT TwilightII 3
EXT TwilightII 3

Manual Settings Control

The TwilightII settings can be manually controlled from a dialog window.
This window is called up from a script using the following command:

EXT TwilightII

Note that no function number is given for this command. If the SHR 640 desktop is not currently showing, this command will do nothing.

Select the required setting by using the buttons:

[ ] Inactivate Twilight II while using Spectrum
While online or when running a script:
(•) Use background blanking
( ) Inactivate Twilight II

The setting will be saved to disk when the Save button is pressed.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop
Updated by: Ewen Wannop

December 8th 1996
July 21st 2003

This technical note describes the WindowMgr XCMD v1.2 and later

The WindowMgr XCMD

Using the WindowMgr XCMD scripts can manipulate windows and dialogs directly from a script. The window definition, and any other controls or menu definitions, must be supplied as resource templates. These resources are then loaded by the script and interaction can take place either directly from a WindowMgr Function call, or indirectly from Spectrum's own TaskMaster routines.

The controls within the window, and the window itself, can be interrogated and controlled by a series of WindowMgr calls. Not every possible control or menu function has been enabled. It is expected that the script writer will use a resource editor to build the appropriate resources rather than constructing controls or menus entirely from this XCMD. If necessary, resources can be built, or manipulated, directly from the ResEdit XCMD.

The WindowMgr XCMD supports inbound 'Inter XCMD Communication'. This allows faster responses where one XCMD wishes to call another through direct IPC calls, rather than through a Spectrum controlled script. See the main XCMD documentation for more details.

The WindowMgr XCMD will only load and be active in versions of Spectrum later than v2.1. Hierarchic is required for Spectrum to start up, so it will always be available for those commands requiring it.

A knowledge of the Control Manager and Window Manager is required to use some of the WindowMgr calls. While every effort is made to check the validity of passed values, setting an inappropriate or incorrect value for many of these calls might have unpredictable and possibly unforseen results, or even cause Spectrum to crash! The logical flow of a script must also be carefully worked out to make windows behave as a User would expect.
Test your scripts thoroughly before giving them to others to use.

If you are in doubt... Don't use this XCMD!

General Notes

Command summary:

0	GetVersion	See if WindowMgr is installed
1	LoadWindow	Create a new window
2	DisposeWindow	Closes an open window
3	ProcessWindow	Handle events in the window
4	ActivateWindow	Makes a window active (bring to front)
5	DeactivateWindow	Makes a window inactive (send to back)
6	ShowWindow	Makes a window visible
7	HideWindow	Makes a window invisible
8	SizeWindow	Resize and/or move window

9	SetWindowTitle	Set/Change the window title
10	GetWindowStatus	Check the state of a window
11	MoveControl	Moves a control within a window
12	DimControls	Deactivate control(s) in a window
13	UndimControls	Activate control(s) in a window
14	GetCtlValue	Gets the value field of a control
15	SetCtlValue	Sets the value field of a control
16	HideControls	Makes some control(s) invisible
17	ShowControls	Makes some control(s) visible
18	GetCtlTtitle	Gets the ctlData field of a control
19	SetCtlTitle	Sets the ctlData field of a control
20	GetLEText	Gets the text of an LineEdit control
21	SetLEText	Sets the text of an LineEdit control
22	GetTEText	Gets the text & style of a TextEdit control
23	SetTEText	Sets the text & style of a TextEdit control
24	GetCtlRect	Gets the current rectangle of a control
25	SetCtlRect	Sets the current rectangle of a control
26	DisposeCtl	Removes a control from a window
27	NewCtl	Inserts a control into a window
28	LoadListFile	Builds a list from a file template
29	SetHitAction	Sets the response to unclaimed control hits
30	InsertMenuBar	Inserts a menu bar into a window
31	SetNewSysMenuBar	Replaces the system Menu bar
32	CloseNewSysMenuBar	Returns to Spectrum Menu Bar
33	DisableMenu	Disables or dims a Menu
34	EnableMenu	Enables or undims a Menu
35	SetMItemFlags	Checks and hilites menu items
36	SetMItemStyle	Sets font attributes for menu item
37	InsertMItem	Inserts a menu item into a menu
38	DeleteMItem	Deletes a menu item
39	SetMItemIcon	Sets an icon to a menu item
40	SetPopItemFlags	Checks and hilites popup menu items
41	SetPopItemStyle	Sets font attributes for popup menu item
42	InsertPopItem	Inserts a menu item into a popup menu
43	DeletePopItem	Deletes a popup menu item
44	SetPopItemIcon	Sets an icon to a popup menu item
45	SetPopPathItem	Inserts a Pathname into a popup menu
46	FindFrontWindow	Returns the GrafPort of the front window
47	InsertMItem2	Inserts a menu item by string into a menu
48	InsertPopItem2	Inserts a menu item by string into a popup menu
49	GetMItemFlags	Returns the current state of a Menu Item
50	GetWindRect	Returns the current global window rectangle
51	InsertMenu	Inserts a new menu into a Menu Bar
52	DeleteMenu	Deletes a menu from a Menu Bar
53	InsertHierarchic Menu	Inserts a menu to be controlled by Hierarchic
54	DeleteHierarchic Menu	Deletes a menu controlled by Hierarchic
55	SetMItemFlags2	Checks and hilites items in a Hierarchic submenu
56	SetMItemStyle2	Sets font attributes for items in a Hierarchic submenu
57	GetListValue	Gets a selected item in a list and resets the item
58	SetWindowFont	Sets the Font that will be used for a New Window

Numbers can be given in both Decimal or Hexadecimal form. If the number is given in Hexadecimal form, the number must be preceeded by a '$'. Note that this will require a double '$$' within the script command. Returned numbers will be in either Decimal or Hexadecimal format. If in Hexadecimal, the number will be preceeded by a '$'. Use the HodgePodge number routines to convert from Decimal to Hexadecimal and vice versa as needed.

Where a ResourceID is asked for, this may be given as a Hexadecimal or Decimal number as the "Resource Name". The resource name must be given within the current Quote characters, and must be exactly the same upper/lower case as the resource.

Only 16 windows may be opened by the WindowMgr at any one time. All windows opened by the XCMD, and custom menus, will be closed when the script stops.

Note that except for Function 0, all Functions will fail if the 640 SHR desktop, or the Spectrum SHR display, is not currently showing.

In building window templates for use by the WindowMgr XCMD, it is important to remember that these windows will at times be open and active under the main TaskMaster loop within Spectrum. File holding resource templates must be opened within the script prior to calling Function 1, and must remain open for all the WindowMgr calls while that script is active, or until the window referenced by that file is closed. Note that any resource files opened by a script will be automatically closed when the script stops. Failure to keep an active resource file open will result in a "Resource not found" system crash when the WindowMgr XCMD tries to use the window resource! To avoid conflicts with Spectrum's own resources, you must ensure that all resources within your file have a unique ID from the following ranges:

- XCMDs can safely use resource ID's with values greater than those listed, but remember that these controls will also reference other resources which might conflict with those of Spectrum.
- Use pString resource IDs greater than $11000
  MenuItem resource IDs greater than $700
  LETextBox resource IDs greater than $50
  Window resource IDs greater than $1000
  Alert resource IDs greater than $20
- If in doubt about a resource ID value, check the resource fork of Spectrum to see if your resource will conflict with any of the resource IDs belonging to Spectrum itself.
  But be aware that programs like TransProg III may have inserted their own menu items into their own menus in your menubar. The item IDs may conflict with the IDs you have chosen for your menus, and so the wrong item may become dimmed or otherwise changed. There is no easy solution to this, but if you see odd things happening with another applications menus, then try a different range of IDs for yours.
- Note that it is only the conflicting resource IDs that can cause problems. The control ID of a control is not a problem, and can normally be set to anything you like. It is this control ID, in conjunction with the GrafPort of the target window, that you will use for the WindowMgr calls. If you wish to use Function 29, or let Spectrum's TaskMaster loop report 'unclaimed hits', you must make sure that control IDs are greater than $7000 and less than $FFFF.

The WindowMgr XCMD supports XCMD Intercommunication. This allows XCMDs to directly interface with the WindowMgr and have responses returned to them rather than Spectrum.

Hierarchic Menus

Hierarchic menus require special consideration. Only 16 menus can be inserted by the WindowMgr at any one time. If you have not already deleted any inserted menus, they will be deleted when your script stops. You will need to refer to the programming notes for Hierarchic for details of how to construct Hierarchic sub-menus.

Do not attempt to delete a Spectrum menu, or insert items into a Spectrum menu that is already being controlled by Hierarchic from Spectrum itself. If you do, you may well cause Spectrum to hang or to fail to quit correctly.

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT WindowMgr 0 {VarName}
EXT WM 0 {VarName}
```

The VarName, if present, will return the version number of the XCMD.

Note that from version 2.7 of the WindowMgr XCMD, WM and WM2 can be used instead of WindowMgr and WindowMgr2.

Function = 1 (LoadWindow)

Function 1 loads a window from a resource template in an open resource file:

```
EXT WindowMgr 1 <resourceID> <VarName>
EXT WindowMgr 1 $00008001 GrafPort
EXT WindowMgr 1 1 GrafPort
```

Loads a window of resource type $800E from an open resource file. VarName will return a Pointer to the window's GrafPort. This Pointer is then used for all subsquent WindowMgr calls. Only 16 windows can be opened at any one time. All windows will be closed when the script stops.

The window will display immediately. If you wish to have a delayed display, set the Visible bit in the Window definition to Off. The window will then stay invisible till called with Function 6, ShowWindow.

The Failed flag is set if the resource cannot be loaded, or any required parameters are missing.

Function = 2 (DisposeWindow)

Function 2 closes an open window:

```
EXT WindowMgr 2 <GrafPort>
EXT WindowMgr 2 $$0001100A
```

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 3 (ProcessWindow)

Function 3 processes events within a window:

```
EXT WindowMgr 3 <GrafPort> <Action> <VarName1> {VarName2}
EXT WindowMgr 3 $$0001100A $$0000000B GenResult {SpecificResult}
```

Where Action is a 32 bit flag and controls the results passed back to Spectrum:

Bit 0 = 1 Active
1 = 1 First hit on Control only
2 = 1 CloseBox hits
3 = 1 Simple Button hits
4 = 1 Check Box hits
5 = 1 Icon Button hits
6 = 1 Line Edit hits
7 = 1 List Control hits
8 = 1 Picture Control hits
9 = 1 Pop-up Control hits
10 = 1 Radio Button hits
11 = 1 Scroll Bar hits
12 = 1 Size Box hits
13 = 1 Static Text hits
14 = 1 Text Edit Control hits
15 = 1 Thermometer Control hits
16 = 1 Rectangle Control hits
17 = 1 All key presses
18 = 1 Escape presses

Note: Some of these bit values are mutually exclusive. Bit 0 must be set for any action to take place. It will probably be easier to set the correct values if you pass a Hexadecimal number.

Returned values:

GenResult	*SpecificResult*	*Result Format*	*Description*
0	N/A	N/A	CloseBox pressed
1	ControlID	Hexadecimal	A control was hit
2	ASCII code	Decimal	Unknown key pressed
3	MenuID	Hexadecimal	Menu Selected

Note: When a window has been created by the WindowMgr XCMD, it is active, and to the front. During the normal execution of a script, the main TaskMaster loop within Spectrum may be called. This can result in controls within the window being processed by Spectrum before you have a chance to call Function 3. As long as the control IDs within your window respect the defined range of IDs, they will be reported to XCMDs as Unclaimed Hits. The WindowMgr can then check any reported hits are for its own windows. If they belong to a WindowMgr window, the hits will be stored, and returned as a normal hit event for Function 3 when it is called. This in effect allows the use of non-modal dialogs from within a suitable script, but see Function 29 for a better way of handling these dialogs. During a Function 3 call, windows will behave as modal dialogs.

ScrollBars are handled in a special way. When paging the thumb is moved by the viewsize, and when using the arrow buttons by a 1/20th of viewsize. You can then retrieve the new thumb position to see how far you have scrolled. This should suit most situations.

The GrowBox is also treated specially. If you use a GrowBox you must set bit 0 of flag so that TaskMaster will call GrowWindow and SizeWindow to change the size of your window. Be careful with your win dow design if you use a GrowBox, the GrowBox will move when the window is zoomed or sized and you might find that the box is covered up by other controls.

If the CloseBox is pressed while in the Spectrum TaskMaster loop, the window will be closed by Spectrum. Spectrum reports to the WindowMgr it has closed the window, so that window can be marked as being closed. Subsequent calls using that GrafPort will fail as a result. You can use Function 10 at any time to monitor the status of a window to see if it is still open or not.

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 4 (ActivateWindow)

Function 4 activates an inactive window and brings it to the front:

```
EXT WindowMgr  4 <GrafPort>
EXT WindowMgr  4 $$0001100A
```

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 5 (DeactivateWindow)

Function 5 deactivates a window and sends it to the back:

```
EXT WindowMgr  5 <GrafPort>
EXT WindowMgr  5 $$0001100A
```

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 6 (ShowWindow)

Function 6 makes a window visible if it is not already visible:

```
EXT WindowMgr  6 <GrafPort>
EXT WindowMgr  6 $$0001100A
```

Note that if the window was behind another window when this call is made, the window will not be brought to the front. You may also need to use Function 4 to bring the window to the front and therefore make it visible.

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 7 (HideWindow)

Function 7 makes a window invisible if it is not already invisible:

```
EXT WindowMgr  7 <GrafPort>
EXT WindowMgr  7 $$0001100A
```

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 8 (SizeWindow)

Function 8 resizes and moves a window:

```
EXT WindowMgr  8 <GrafPort> <Left> <Right> <Top> <Bottom>
EXT WindowMgr  8 $$0001100A 10 630 20 190
```

Note that these values are global values to the screen, 0,0 being top left and 200,640 being bottom right. If you move the window to near the top of the screen, part of it may be obscured by the System Menu Bar. You should allow for this by placing it at a suitable distance from the top of the screen.

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 9 (SetWindowTitle)

Function 9 sets or changes the title bar of the window:

```
EXT WindowMgr  9 <GrafPort> “TitleString”
EXT WindowMgr  9 $$0001100A “This is a new Title”
```

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 10 (GetWindowStatus)

Function 10 gets the status of the current window:

```
EXT WindowMgr  10 <GrafPort>
EXT WindowMgr  10 $$0001100A
```

This call will fail if the window has been closed either by Spectrum or the WindowMgr.

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 11 (MoveControl)

Function 11 moves a control within a window:

```
EXT WindowMgr  11 <GrafPort> <ControlID> <Left> <Top>
EXT WindowMgr  11 $$0001100A $$7001 20 30
```

Note that these values are local values within the target window. Take care to keep the control within the window if you wish it to be visible.

The Failed flag is set if the GrafPort is not valid, the control does not exist, or any required parameters are missing.

Function = 12 (DimControls)

Function 12 dims control(s) within a window:

```
EXT WindowMgr  12 <GrafPort> <Control1ID> {Control2ID} {Control3ID} {etc.}
EXT WindowMgr  12 $$0001100A $$7001 {$$7002} {$$7003} {etc.}
```

As many control IDs as can fit on the command line can be supplied. Note that when dimmed, some controls are only made inactive, rather then greyed out as is usual.

The Failed flag is set if the GrafPort is not valid, a control does not exist, or any required parameters are missing.

Function = 13 (UndimControls)

Function 13 undims control(s) within a window:

```
EXT WindowMgr  13 <GrafPort> <Control1ID> {Control2ID} {Control3ID} {etc.}
EXT WindowMgr  13 $$0001100A $$7001 {$$7002} {$$7003} {etc.}
```

As many control IDs as can fit on the command line can be supplied.

The Failed flag is set if the GrafPort is not valid, a control does not exist, or any required parameters are missing.

Function = 14 (GetCtlValue)

Function 14 gets the contents of the ctlValue field of a control:

```
EXT WindowMgr  14 <GrafPort> <ControlID> <VarName>
EXT WindowMgr  14 $$0001100A $$7001 ctlValue
```

Not all controls use the ctlValue field and if so will set the Failed flag and return an empty VarName. The actual value returned will depend on the kind of control you are interrogating.

Valid controls and returned ctlValues are:

CheckControl	1 = Control Checked
ListControl	Item in list selected, range 1 to list size.
PopupControl	ID of Menu Item selected
RadioControl	1 = Selected
ScrollBar	Thumb position, 0 to RangeSize
StatText	Length of text
TextEdit	TEHandle in format ~E01234
Thermometer	Position of Mercury

Note that only the first item selected is reported for the ListControl. If you wish to use multiple selection in a list, you should build a window using the more powerful Lister XCMD.

The Failed flag is set if the GrafPort is not valid, the control does not exist, or any required parameters are missing.

Function = 15 (SetCtlValue)

Function 15 sets the contents of the ctlValue field of a control:

```
EXT WindowMgr  15 <GrafPort> <ControlID> <NewValue>
EXT WindowMgr  15 $$0001100A $$7001 1
```

Not all controls use the ctlValue field and so may not respond to this call. If the control is known to ignore this field, the call will set the Failed flag. The WindowMgr will try to make sure that an inappropriate value is not applied to the ctlValue field of a control, to avoid unpredictable results, but ultimately is up to your script to be careful to use valid values while using this command.

Valid controls are CheckControl, ListControl, PopupControl, RadioControl, ScrollBar, StatText and Thermometer. See Function 14 for more details.

The Failed flag is set if the GrafPort is not valid, the control does not exist, or any required parameters are missing.

Function = 16 (HideControls)

Function 16 dims or makes inactive control(s) within a window:

```
EXT WindowMgr  16 <GrafPort> <Control1ID> {Control2ID} {Control3ID} {etc.}
EXT WindowMgr  16 $$0001100A $$7001 {$$7002} {$$7003} {etc.}
```

As many control IDs as can fit on the command line can be supplied.

The Failed flag is set if the GrafPort is not valid, a control does not exist, or any required parameters are missing.

Function = 17 (ShowControls)

Function 17 undims or makes active control(s) within a window:

```
EXT WindowMgr  17 <GrafPort> <Control1ID> {Control2ID} {Control3ID} {etc.}
EXT WindowMgr  17 $$0001100A $$7001 {$$7002} {$$7003} {etc.}
```

As many control IDs as can fit on the command line can be supplied.

The Failed flag is set if the GrafPort is not valid, a control does not exist, or any required parameters are missing.

Function = 18 (GetCtlTitle)

Function 18 gets the contents of the ctlData field of a control:

```
EXT WindowMgr  18 <GrafPort> <ControlID> <VarName>
EXT WindowMgr  18 $$0001100A $$7001 ctlTitle
```

Not all controls use the ctlData field and so may not respond to this call. If the control is known to ignore this field, the call will set the Failed flag and return an empty VarName. Some calls will return a title or name string, and others will return two values as a hexadecimal longword. A simple check to see if a '$' is present at the start of the returned variable will indicate if the returned value is a string or a longword value.

Valid controls are:

SimpleButton	Title String
CheckControl	Title String
IconButton	Title String
ListControl	Lo Word viewSize, Hi Word listSize
RadioControl	Title String
ScrollBar	Lo Word viewSize, Hi Word dataSize
StatText	Title String
Thermometer	Scale

The Failed flag is set if the GrafPort is not valid, the control does not exist, or any required parameters are missing.

Function = 19 (SetCtlTitle)

Function 19 sets the contents of the ctlData field of a control:

```
EXT WindowMgr  19 <GrafPort> <ControlID> "String"
EXT WindowMgr  19 <GrafPort> <ControlID> <LongWordValue>
EXT WindowMgr  19 $$0001100A $$7001 "New string"
EXT WindowMgr  19 $$0001100A $$7001  $$00xx00yy
```

Not all controls use the ctlData field and so may not respond to this call. If the control is known to ignore this field, the call will set the Failed flag. Some controls require a string as input, others require a longword value as input. Refer to the Control Manager documentation for the settings of these longword values.

Valid controls are SimpleButton, CheckControl, IconButton, RadioControl, ScrollBar, StatText and Thermometer. See Function 18 for more details.

The Failed flag is set if the GrafPort is not valid, the control does not exist, or any required parameters are missing.

Function = 20 (GetLEText)

Function 20 gets the contents of a LineEdit control:

```
EXT WindowMgr  20 <GrafPort> <ControlID> <VarName>
EXT WindowMgr  20 $$0001100A $$7001 theResult
```

The contents of the LineEdit control will be returned in the variable.

The Failed flag is set if the GrafPort is not valid, the control is not a LineEdit control, does not exist, or any required parameters are missing.

Function = 21 (SetLEText)

Function 21 sets the contents of a LineEdit control:

```
EXT WindowMgr  21 <GrafPort> <ControlID> “String” {Start} {End}
EXT WindowMgr  21 $$0001100A $$7001 “New text for LineEdit control” {0} {256}
```

Sets the contents of the LineEdit control. Note that the amount of text that can be set, depends on the size of the maximum length of input line that was created within the LineEdit control resource.

The optional Start and End can be given to set the selection within the displayed record. The range is 0-256 and the Start must be equal to or less than the End. If they are equal, then the cursor will flahs at the insertion point. If the optional Start and End are not given, the entire text will be selected.

The Failed flag is set if the GrafPort is not valid, the control is not a LineEdit control, the control does not exist, or any required parameters are missing.

Function = 22 (GetTEText)

Function 22 gets the contents of a TextEdit control:

```
EXT WindowMgr  22 <GrafPort> <ControlID> <TargetEditor> {Start} {End}
EXT WindowMgr  22 $$0001100A $$7001 $EditorHandle Start End
EXT WindowMgr  22 $$0001100A $$7001 ~E01234
```

The TargetEditor must be given as a ScriptEditor Handle, and must already exist.

The optional Start and End variables may be given to return the current selection points of the text.

The Failed flag is set if the GrafPort is not valid, the control is not a TextEdit control, the control does not exist, or any required parameters are missing.

Function = 23 (SetTEText)

Function 23 sets the contents of a TextEdit control:

```
EXT WindowMgr  23 <GrafPort> <ControlID> <SourceEditor> {Start} {End}
EXT WindowMgr  23 $$0001100A $$7001 $EditorHandle 0 256
EXT WindowMgr  23 $$0001100A $$7001 ~E01234
```

The SourceEditor must be given as a ScriptEditor Handle, and must already exist.

The optional Start and End positions may be given to set the current selection of the text. If these are not given, the selection point will be set to the start of the text.

The Failed flag is set if the GrafPort is not valid, the control is not a TextEdit control, the control does not exist, or any required parameters are missing.

Function = 24 (GetCtlRect)

Function 24 gets the current rectangle of a control:

```
EXT WindowMgr  24 <GrafPort> <ControlID> <VarName>
EXT WindowMgr  24 $$0001100A $$7001 CtlRectangle
```

The rectangle is returned in the VarName in the format 'Left,Right,Top,Bottom'. Each value is in decimal and spaced by a comma. You can use the substring script commands to extract each value. The rectangle can be used to find where within a window the control is, and use this information as input for Functions 11 and 25.

The Failed flag is set if the GrafPort is not valid, the control does not exist, or any required parameters are missing.

Function = 25 (SetCtlRect)

Function 25 sets the current rectangle of a control:

```
EXT WindowMgr  25 <GrafPort> <ControlID> <Left> <Right> <Top> <Bottom>
EXT WindowMgr  25 $$0001100A $$7001 10 110 20 170
EXT WindowMgr  25 $$0001100A $$7001 10,110,20,170
```

The effect of changing the rectangle of a control, will depend on what kind of control the target control is. In some cases it will change the active 'hit' area, and in other cases the actual display size or frame. Refer to the Control Manager documentation for further information.

The Failed flag is set if the GrafPort is not valid, the control does not exist, or any required parameters are missing.

Function = 26 (DisposeCtl)

Function 26 removes a control from a window:

```
EXT WindowMgr  26 <GrafPort> <ControlID>
EXT WindowMgr  26 $$0001100A $$7001
```

The Failed flag is set if the GrafPort is not valid, or any required parameters are missing.

Function = 27 (NewCtl)

Function 27 adds a new control to a window:

```
EXT WindowMgr  27 <GrafPort> <resourceID>
EXT WindowMgr  27 $$0001100A $$00008002
```

Adds a control from an open resource file to a window.

The Failed flag is set if the GrafPort is not valid, the resource could not be found, or any required parameters are missing.

Function = 28 (LoadListFile)

Function 28 builds the contents of a list control:

```
EXT WindowMgr  28 <GrafPort> <ControlID> ":Path:FileName"
EXT WindowMgr  28 $$0001100A $$7001 ":Ram5:ResourceFile"
```

The List control must be of a suitable size to handle the data in the list file. A list file follows the specifications of the Lister XCMD files, and you can use the output files from the Lister XCMD as an input to this function. This allows dynamic control of your lists from within a script. The format is:

```
<Index> "String" {Value} {Selected} <CR>
```

The {Value} field will be ignored, but to allow Lister files to be used, must be present if you wish to use the optional {Selected} field to preselect a list item.

Any existing list will be replaced with the new file contents. You can use the other WindowMgr Functions to examine the field sizes of a List control. Note that the current settings of the QuoteCharacter will be used when loading the List file into the control. You should be aware of the Quote character used in any source files, and set the current QuoteCharacter if necessary.

The Failed flag is set if the GrafPort is not valid, the file could not be found, or any required parameters are missing.

Function = 29 (SetHitAction)

Function 29 sets the action to be taken on unclaimed hits:

```
EXT WindowMgr  29 <Action> <Window> <Control>
EXT WindowMgr  29 $$0000000B WindowHit ControlHit
EXT WindowMgr  29 0
```

The two variables are optional and are ignored if Bit 0 of Action is not set.
Action is a 32 bit flag and controls the results passed back to Spectrum:

```
Bit  0 = 1   Active (set to 0 to inactivate)
     1 = 1   First hit on Control only
     2 = 1   CloseBox hits
     3 = 1   Simple Button hits
     4 = 1   Check Box hits
     5 = 1   Icon Button hits
     6 = 1   Line Edit hits
     7 = 1   List Control hits
     8 = 1   Picture Control hits
     9 = 1   Pop-up Control hits
     10 = 1  Radio Button hits
     11 = 1  Scroll Bar hits
     12 = 1  Size Box hits
     13 = 1  Static Text hits
     14 = 1  Text Edit Control hits
     15 = 1  Thermometer Control hits
     16 = 1  Rectangle Control hits
     17 = 1  Double Click List Items
     18 = 1  Not applicable
```

Note: Some of these bit values are mutually exclusive. Bit 0 must be set for any action to take place. It will probably be easier to set the correct values if you pass a Hexadecimal number.

When Spectrum sees a control hit in a window, and the control ID is greater than $7000, it sends an IPC message to say that there has been a hit on an unclaimed control. XCMDs picking up this message should check that the control belongs to them before taking any further action. If it does belong to them, they should acknowledge and respond appropriately.

At the start of a script, Function 29 is inactive and any unclaimed hits will be sent to Function 3. If Function 29 is turned on by setting State =1, unclaimed hits will be reported to Spectrum as script commands instead:

```
Set Variable WindowHit $00012345
Set Variable ControlHit $00001234
```

Clicking on the Close box of a window is a special case and will be reported as a control with an ID of $00000000. This allows scripts to monitor the close box and close the window as and when appropriate.

When using List controls, you must set Bit 7. If you also set Bit 17, then Bit 1 will be ignored, and only double-clicks on a list item will be reported.

An IPC message is also be sent to 'wake up' Spectrum from a 'Suspend Script' command. By putting a script into suspension, and then waiting for any hits on controls within a WindowMgr window, will ensure that these script commands will be seen immediately after they are sent.

Note that inserted MenuBars, ScrollBars and GrowBoxes will not respond to this Function.

The Failed flag is set if any required parameters are missing.

Function = 30 (InsertMenuBar)

Function 30 inserts or replaces a menubar into a window:

```
EXT WindowMgr  30 <GrafPort> <resourceID>
EXT WindowMgr  30 $$0001100A $$00008003
```

Note that there are some limitations with using Menu Bars within WindowMgr windows. They can only be selected when using Function 3 when the window is acting as a modal dialog. They will not be selectable when using the Window as a non-modal dialog, or when Spectrum's TaskMaster has control.

The Failed flag is set if the GrafPort is not valid, the resource could not be found, or any required parameters are missing.

Function = 31 (SetNewSysMenubar)

Function 31 replaces the Spectrum system Menubar with a custom MenuBar:

```
EXT WindowMgr  31  <Method> <resourceID> <MenuItemID> {Title}
EXT WindowMgr  31 0 $$0000800F MenuID MenuTitle
```

Where Method = 0 for Spectrum to respond normally to standard MenuKey selections even if they are not within the displayed MenuBar. This is normal practice within Spectrum to allow displays that do not have a menu bar to operate correctly. Set Method = 1 to let Spectrum respond to Menu Bar equivalents only. You should be careful when using this command. If you use a display that does not show a Menu bar, you should make sure that you have the OA-W equivalent active, so you can at least close the window if necessary. Another way would be to have a Menu equivalent that will allow the sc ript to stop and have Spectrum's menu bar to be restored.

The <Method> value will only apply if you are replacing the Spectrum MenuBar itself. It will have no effect on Spectrum if you are replacing a custom MenuBar you have already installed.

Selections in your custom menus are reported to XCMDs through IPC calls. Whenever you have installed a MenuBar, the WindowMgr XCMD will look for and trap ALL reported Menu hits greater than $200. You should use the Suspend Script command to get immediate response from these hits. The WindowMgr then inserts the results into the script stream and wakes up Spectrum. See Function 29 for more details on how this is done. The MenuBar selections of your custom menus are not accessible through Function 3.

When the new MenuBar has been installed, you will lose access to the normal Spectrum menus and Open Apple commands. Some of these OA commands are trapped within Spectrum so that they are accessible from custom displays, you should not use any Key equivalents of those commands.

Generally you should avoid any IDs that could clash with Spectrum itself. It is safe to use IDs greater than $200 and less then $7FE. If you use any of Spectrum's own ID's, then be careful which ones you choose. Spectrum will act on these hits just as if its own menus were displayed. With care, this can give you access to some of Spectrum's features. It is permissable to use the special menu items, as these are trapped normally by TaskMaster.

Do not use Spectrum's own 'Stop Script' menu item. By doing so you will bypass your own routines to close any windows that you may have opened. This will result in Spectrum being unable to close the linked resource files to these windows. This could cause problems later if these files are referenced again. You should handle stopping the script yourself so you can close all the windows you have opened. When you then issue 'Stop Script', the resource files will be correctly closed.

It is advisable to use the script commands 'Save Settings' and 'Restore Settings' at the start and end of your scripts. This will ensure that the Spectrum menus are restored correctly. There may still be problems with some menu items remaining dimmed after you use a new System MenuBar. This will be addressed in Spectrum 2.2.

The mandatory MenuItemID variable, will return the ID of any MenuItem selected that is greater than $200. The optional MenuTitle variable will return the text of the MenuItem selected. You might find this a more suitable way of matching results than the MenuID itself.

The Failed flag is set if any required parameters are missing.

Function = 32 (CloseNewSysMenuBar)

Function 32 closes the custom MenuBar and restores the Spectrum MenuBar:

```
EXT WindowMgr  32
```

If you are using Spectrum 2.1 the current dimmed setting of all menu items in the Spectrum Menus are remembered when you draw your first MenuBar with Function 31. When you close your final MenuBar, restoring the Spectrum MenuBar, these settings are returned. Be aware that this might still cause some problems if you have used any of Spectrum's own Menu Items in your menus and the settings have changed. It would be wise to issue a 'Clear Desktop' command before you open any of your own SystemMenuBars. This should avoid such problems occurring.

Spectrum 2.2 and later is 'smart' and will restore the state of the Menu Items itself.

The Failed flag is set if any required parameters are missing.

Function = 33 (DisableMenu)

Function 33 disables a Menu:

```
EXT WindowMgr 33 <GrafPort> <MenuID> {MenuID2} {MenuID3} {etc}
EXT WindowMgr 33 $$0001100A $$0001 {$$0002} {$$0003} {etc}
```

Use a value of '0' for GrafPort when referring to the System Menu bar.
As many Menu IDs as can fit on the command line can be supplied.

The Failed flag is set if any required parameters are missing.

Function = 34 (EnableMenu)

Function 34 enables a Menu:

```
EXT WindowMgr 34 <GrafPort> <MenuID> {MenuID2} {MenuID3} {etc}
EXT WindowMgr 34 $$0001100A $$0001 {$$0002} {$$0003} {etc}
```

Use a value of '0' for GrafPort when referring to the System Menu bar.
As many Menu IDs as can fit on the command line can be supplied.

The Failed flag is set if any required parameters are missing.

Function = 35 (SetMItemFlags)

Function 35 checks, unchecks, enables or disables Menu Items:

```
EXT WindowMgr 35 <GrafPort> <Action> <MenuItemID> {CheckMark}
EXT WindowMgr 35 $$0001100A 0 $$0301 {94}
```

Use a value of '0' for GrafPort when referring to the System Menu bar.

Where Action:

0	Clear check mark, remove underline and enable MenuItem
1	Check MenuItem
2	Uncheck MenuItem
3	Enable MenuItem
4	Disable MenuItem
5	Underline MenuItem
6	Don't underline MenuItem

You may use combinations of these additional flags for Action, the number can be given in Hex or decimal:

$0020	Use XOR highlighting
$0040	Underline Item
$0080	Disable menu item
$FF7F	Enable menu item
$FFBF	Don't underline item
$FFDF	Use redraw highlighting

If the optional {CheckMark} is given, this will be used instead of the plain Check for Action 1. The optional checkmark does not apply to the additional flags.

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 36 (SetMItemStyle)

Function 36 sets the style of a Menu Item:

```
EXT WindowMgr 36 <GrafPort> <Action> <MenuItemID>
EXT WindowMgr 36 $$0001100A 16 $$0301
```

Where Action is a bitwise value that sets the font attributes:

```
Bit 0 = 1 make Bold
Bit 1 = 1 make Italic
Bit 2 = 1 make Underline
Bit 3 = 1 make Outline
Bit 4 = 1 make Shadow
```

Passing a value of 0 for Action, will reset to plain text.

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 37 (InsertMItem)

Function 37 inserts a Menu Item into a target Menu:

```
EXT WindowMgr 37 <GrafPort> <resourceID> <MenuID> {InsertAfterID}
EXT WindowMgr 37 $$0001100A $$00012345 $$0021 0
```

The MenuID is the ID of the Menu to insert the item into. The optional InsertAfterID is 0 to insert at the front, $FFFF to insert at the end, or the actual MenuItemID to insert the new item after. If the optional value is not present, it defaults to insert at the end.

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 38 (DeleteMItem)

Function 38 deletes a Menu Item:

```
EXT WindowMgr 38 <GrafPort> <MenuItemID> <MenuID>
EXT WindowMgr 38 $$0001100A $$0301 $$0021
```

Note that the MenuID must be given so that the menu sizes will be calculated correctly after deleting the Item.

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 39 (SetMItemIcon)

Function 39 sets an Icon to a Menu Item:

```
EXT WindowMgr 39 <GrafPort> <resourceID> <MenuItemID> <MenuID>
EXT WindowMgr 39 0 $$00012345 $$0301 $$0021
```

Note that the MenuID must be given so that the menu sizes will be calculated correctly after inserting the Icon.

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 40 (SetPopItemFlags)

Function 40 checks, unchecks, enables or disables popup Menu Items:

```
EXT WindowMgr 40 <GrafPort> <Action> <PopResourceID> <MenuItemID>
                                                          {CheckMark}
EXT WindowMgr 40 $$0001100A 0 $$00002001 $$0301 {94}
```

Note that for all the popup functions, the popup resource ID must be given so that the menu can be redrawn correctly.

See Function 35 for further details of the Action flags.

Function = 41 (SetPopItemStyle)

Function 41 sets the style of a popup Menu Item:

```
EXT WindowMgr 41 <GrafPort> <Action> <PopResourceID> <MenuItemID>
EXT WindowMgr 41 $$0001100A 16 $$00002001 $$0301
```

See Function 36 for further details.

Function = 42 (InsertPopItem)

Function 42 inserts a Menu Item into a target popup Menu:

```
EXT WindowMgr 42 <GrafPort> <resourceID> <PopResourceID>  <MenuID>
                                                          {InsertAfterID}
EXT WindowMgr 42 $$0001100A $$00012345 $$00002001 $$0021 0
```

See Function 37 for further details.

Function = 43 (DeletePopItem)

Function 43 deletes a popup Menu Item:

```
EXT WindowMgr 43 <GrafPort> <PopResourceID> <MenuItemID> <MenuID>
EXT WindowMgr 43 $$0001100A $$00002001 $$0301 $$0021
```

See Function 38 for further details.

Function = 44 (SetPopItemIcon)

Function 44 sets an Icon to a popup Menu Item:

```
EXT WindowMgr 44 <GrafPort> <resourceID> <PopResourceID> <MenuItemID>
                                                                <MenuID>
EXT WindowMgr 44 $$0001100A $$00012345 $$00002001 $$0301 $$0021
```

See Function 39 for further details.

Function = 45 (SetPopPathItem)

Function 45 inserts a Pathname into a popup Menu:

```
EXT WindowMgr 45 <GrafPort> ":Path:File" <PopResourceID> <MenuItemID>
                                  <StartID> <Direction> {InsertAfterID} {Result}
EXT WindowMgr 45 $$0001100A ":Hard:One" $$00002001 $$00012345
                                          $$0100 0 {$$00FA} {HighestID}
```

Where Direction = 0 for ascending path and files, and Direction = 1 for descending files and paths. If the optional {InsertAfterID} is not given, the path will be inserted at the top of the menu. The <StartID> is used as the starting ID for the inserted items, and if the optional {HighestID} variable is given, this will return the highest ID used. You should take care to avoid conflicts with existing menu items or Spectrum's menu items.

The Failed flag is set if any of the required parameters are missing or invalid.

Function = 46 (FindFrontWindow)

Function 46 returns the GrafPort of the front window:

```
EXT WindowMgr 46 <Result>
EXT WindowMgr 46 TopWindow
```

The Failed flag is set if the window is not a WindowMgr window, or the required parameters are missing.

Function = 47 (InsertMItem2)

Function 47 inserts a Menu Item into a target Menu:

```
EXT WindowMgr 47 <GrafPort> <"String"> <MenuItemID> <MenuID>
                                        {InsertAfterID} {KeyEquivalents}
EXT WindowMgr 47 $$0001100A "MenuItemString" $$0301 $$0021 0
```

The string is the menu description and the MenuItemID is the ID that will be used for this Menu item. The MenuID is the ID of the Menu to insert the item into. The optional InsertAfterID is 0 to insert at the front, $FFFF to insert at the end, or the actual MenuItemID to insert the new item after. If the optional value is not present, it defaults to insert at the end.

The optional KeyEquivalents are two characters representing the OpenApple key presses for this item. The two characters must not be presented in Quotes. This allows the current Quote character to be used as a key equivalent. The first character given will be the one displayed with the Menu item.

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 48 (InsertPopItem2)

Function 48 inserts a Menu Item into a target popup Menu:

```
EXT WindowMgr 48 <GrafPort> <"String"> <MenuItemID> <PopResourceID>
                                    <MenuID> {InsertAfterID} {KeyEquivalents}
EXT WindowMgr 48 $$0001100A "MenuItemString" $$0301 $$00002001 $$0021 0
```

The optional KeyEquivalents are two characters representing the OpenApple key presses for this item. The two characters must not be presented in Quotes. This allows the current Quote character to be used as a key equivalent. The first character given will be the one displayed with the Menu item.

See Function 47 and 37 for further details.

Function = 49 (GetMItemFlags)

Function 49 gets the current Menu Item Flags:

```
EXT WindowMgr 49 <GrafPort> <MenuItemID> <VarName>
EXT WindowMgr 49 $$0001100A $$0301 Value
```

Use a value of ‘0’ for GrafPort when referring to the System Menu bar.

Only the first 8 bits of the returned value are valid. The value can be used for subsequent input to Function 35. Use HodgePodge if you need to EOR the Hex number.

The Failed flag is set if any required parameters are missing.

Function = 50 (GetWindowRect)

Function 50 gets the global rectangle for a window:

```
EXT WindowMgr 50 <GrafPort> <VarName>
EXT WindowMgr 50 $$0001100A Value
```

Returns the global rectangle of the window in decimal values. The format is "Left Right Top Bottom". You can check if a window has moved or has changed size, by making the 'If Equal' comparison on its original position against the current position.

The numbers are given in six character blocks with leading spaces. This will make it easier to check the values by 'SubString' commands if desired. The result of this Function can be passed to Function 8 to reposition the window.

The Failed flag is set if any required parameters are missing.

Function = 51 (InsertMenu)

Function 51 inserts a Menu:

```
EXT WindowMgr 51 <GrafPort> <templateID> {InsertAfterID}
EXT WindowMgr 51 $$0001100A $$0301 $$0020
```

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 52 (DeleteMenu)

Function 52 deletes a Menu:

```
EXT WindowMgr 52 <GrafPort> <MenuID>
EXT WindowMgr 52 $$0001100A $$0021
```

Note: Do not attempt to delete menus that are being controlled by Hierarchic from Spectrum itself.

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 53 (InsertHierarchicMenu)

Function 53 inserts a menu to be controlled by Hierarchic:

```
EXT WindowMgr 53 <GrafPort> <templateID> {InsertAfterID}
EXT WindowMgr 53 $$0001100A $$0301 $$0020
```

The Failed flag is set if the menu is already being controlled by Hierarchic, if the resource could not be found, or any required parameters are missing.

Function = 54 (DeleteHierarchicMenu)

Function 54 removes a menu controlled by Hierarchic:

```
EXT WindowMgr 54 <GrafPort> <MenuID>
EXT WindowMgr 54 $$0001100A $$0021
```

Note: Do not attempt to delete menus that are being controlled by Hierarchic from Spectrum itself.

The Failed flag is set if the menu is not being controlled by Hierarchic, if the resource could not be found, or any required parameters are missing.

Function = 55 (SetMItemFlags2)

Function 55 checks, unchecks, enables or disables Menu Items.

```
EXT WindowMgr 55 <GrafPort> <Action> <HostMenuID> <MenuItemID>
                                                    <SubMenuID> {CheckMark}
EXT WindowMgr 55 $$0001100A 0 $$0001 $$0301 $$0021 {94}
```

The 'HostMenuID' is that of the menu containing the sub-menu.

This function operates in a similar way to Function 35, but must be used for items in Hierarchic sub-menus.

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 56 (SetMItemStyle2)

Function 56 sets the style of a Menu Item.

```
EXT WindowMgr 56 <GrafPort> <Action> <HostMenuID> <MenuItemID>
                                                          <SubMenuID>
EXT WindowMgr 56 $$0001100A 16 $$0001 $$0301 $$0021
```

The 'HostMenuID' is that of the menu containing the sub-menu.

This function operates in a similar way to Function 36, but must be used for items in Hierarchic sub-menus.

The Failed flag is set if the resource could not be found, or any required parameters are missing.

Function = 57 (GetListValue)

Function 57 gets the next selected item of a list control:

```
EXT WindowMgr  57 <GrafPort> <ControlID> <Value> <VarName>
EXT WindowMgr  57 $$0001100A $$7001 0 ctlValue
```

The next item from Value that is selected will be returned. If there are no more items selected, the value returned will be 0. Using a Value of 0, thefirst selected item will be returned. This gives some control over lists attached to windows, but for complete control of a list, use the more powerful Lister XCMD.

You can also select a single item from a list control by using Function 15. This will however deselect any other selected items.

The Failed flag is set if the GrafPort is not valid, the control does not exist, the control is not a list control, or any required parameters are missing.

Function = 58 (SetWindowFont)

Function 58 sets the Font that will be used for a New Window:

```
EXT WindowMgr  58 “FamilyName” <Size>
EXT WindowMgr  57 “Spectrum”
```

<Size> is optional. The normal 8pt Shaston System Font will be used when a New Window is created. You may choose to use another Font to control List displays for instance. The FamilyName is reset to Shaston 8 whenever a script starts.

The Failed flag is set an error occurs or the required parameter is missing.

Spectrum XCMD Technical Notes

Written by: Ewen Wannop
Updated by: Ewen Wannop

May 1995
June 12th 2003

This technical note describes the WorkBench XCMD

The WorkBench XCMD

The WorkBench XCMD allows editing and examination of the resource fork of a file, Tool Call support, GS/OS support, IPC and Message centre calls, Peek and Poke. Functions 1-8 of the WorkBench XCMD are identical to those of the ResEdit XCMD which this XCMD now replaces.

Note: This is a potentially extremely dangerous XCMD, and if used without due care, can damage files on your hard disk. It should only be used with very great caution, and only if the script author is fully conversant with the structure of resources and the resource fork., and the structure of any Tool calls that are made using it. It is advisable only to install or activate this XCMD if you are absolutely sure of the provenance of any scripts you might run. Never run a script that has come from an unknown source if this XCMD is present. Such a script might have all the effects of a dangerous virus. You have been warned!

Function = 0 (get version number)

Function 0 gets the version number, the Failed flag is set if the XCMD is not active or present:

```
EXT WB 0 <VarName>
```

The VarName, if present, will return the version number of the XCMD.

Function = 1 (LoadResource)

Function 1 loads the data from a specified resource from a file:

```
EXT WB 1 ":PathName:FileName" ResourceType ResourceID ScriptEditor
EXT WB 1 ":PathName:FileName" ResourceType "ResName" ScriptEditor
EXT WB 1 ":Hard.Disk:TestFile" $$8006 $$00000001 ~E01234
```

Where: ResourceType = the target resource type
ResourceID = the ID of the target resource

Note that ResourceID can be given either as a number or a "ResName".

If the resource is found, the data block of the resource will be loaded and returned in the ScriptEditor.

If the file does not exist, or is already open, or does not have a resource fork, or the resource does not exist, or the EditorHandle does not exist or is invalid, or any input error occurs the Failed flag will be set.

Function = 2 (SaveResource)

Function 2 saves the data to a specified resource in a file:

EXT WB 2 ":PathName:FileName" ResourceType ResourceID Attributes ScriptEditor EXT WB 2 ":PathName:FileName" ResourceType "ResName" Attributes ScriptEditor EXT WB 2 ":Hard.Disk:TestFile" $$8006 $$00000001 $0000 ~E01234

Where: ResourceType = the target resource type
ResourceID = the ID of the target resource
Attributes = resource attributes

Note that ResourceID can be given either as a number or a "ResName". If a "ResName" is given a unique ID will be generated for the saved resource.

If the file exists or is already open, and has a resource fork, the contents of the ScriptEditor will be saved as a new resource. If a resource of the same type and number already exists, or if the "ResName" already exists, the resource will not be written and the failed flag will be set.

If the file does not exist, or does not have a resource fork, or the EditorHandle does not exist or is invalid, or any input error occurs the Failed flag will be set.

Function = 3 (DeleteResource)

Function 3 deletes a specified resource from a file:

EXT WB 3 ":PathName:FileName" ResourceType ResourceID
EXT WB 3 ":PathName:FileName" ResourceType "ResName"
EXT WB 3 ":Hard.Disk:TestFile" $$8006 $$00000001

Use this command to clear an existing resource before using the SaveResource command.

Note that ResourceID can be given either as a number or a "ResName".

If the file does not exist, or is already open, or does not have a resource fork, or the resource does not exist, or any input error occurs, the Failed flag will be set.

Function = 4 (CreateResourceFork)

Function 4 creates an empty resource fork in a file:

EXT WB 4 ":PathName:FileName" {FileType} {AuxType}
EXT WB 4 ":Hard.Disk:TestFile" {$$FF} {$$0000}

If the optional FileType and AuxType are not present, and the file does not already exist, the filetype will be set $06 and the AuxType to $0000. If the file already exists it is not necesary to provide FileType and AuxType.

The file will be created if it does not already exist.

If the file already has a resource fork, or any input error occurs, the Failed flag will be set.

Function = 5 (GetResourceList)

Function 5 gets a list of all the resources in a file:

EXT WB 5 ":PathName:FileName" ScriptEditor
EXT WB 5 ":Hard.Disk:TestFile" ~E01234

The returned information is in the form of CR delimited records with Hexadecimal values:

ResType	ResID	ResAttributes	ResSize	{ResName}
$0000	$00000000	$0000	$00000000	{NameIfExists}

Note that on a large application this function may take several seconds to execute.

If the file does not exist, or is already open, or does not have a resource fork, or the EditorHandle does not exist or is invalid, or any input error occurs the Failed flag will be set.

Function = 6 (ClearResourceFork)

Function 6 clears an existing resource fork in a file leaving an empty fork:

```
EXT WB 6 ":PathName:FileName"
EXT WB 6 ":Hard.Disk:TestFile"
```

This function will generate an error if there is not a correctly formatted GS/OS resource fork in the target file. It will not delete the resource fork of a Macintosh file.

If the file does not exist, or does not have a resource fork, or any input error occurs, the Failed flag will be set.

Function = 7 (GetResourceName)

Function 7 gets the name of a specified resource from a file:

```
EXT WB 7 ":PathName:FileName" ResourceType ResourceID ResultVariable
EXT WB 7 ":Hard.Disk:TestFile" $$8006 $$00000001 Result
```

If the file does not exist, or is already open, or does not have a resource fork, or the resource does not exist, or any input error occurs the Failed flag will be set.

Function = 8 (SetResourceName)

Function 8 sets the name of a specified resource:

```
EXT WB 8 ":PathName:FileName" ResourceType ResourceID "NewName"
EXT WB 8 ":PathName:FileName" ResourceType "ResName" "NewName"
EXT WB 8 ":Hard.Disk:TestFile" $$8006 $$00000001 "Resource Name"
```

Setting an empty name deletes the name resource for the specified resource.

If the file does not exist, or is already open, or does not have a resource fork, or the resource does not exist, or any input error occurs the Failed flag will be set.

Function 9 = (CompactResourceFork)

Function 9 removes excess space from a resource fork:

```
EXT WB 9 ":PathName:FileName"
EXT WB 9 ":Hard.Disk:TestFile"
```

This function will relocate all the empty blocks in a resource file to the end of the file thus compacting the resource fork.

If the file does not exist, or is already open, or does not have a resource fork, or the resource does not exist, or any input error occurs the Failed flag will be set.

Function 10 = (Peek)

Function 10 will return the contents of a memory location:

EXT WB 10 <Address> <VarName>
EXT WB 10 $$123456 Result

The returned byte is given in decimal, and is the value at the specified location in memory. It makes no check to see if that location actually exists within available memory.

Function 11 = (Poke)

Function 11 will change the contents of a memory location:

EXT WB 11 <Address> <Value>
EXT WB 11 $$123456 $$FF

The supplied value will replace the current byte at the specified location in memory. It makes no check to see if that location actually exists within available memory, is in protected memory or in ROM space. Be careful what you change!

Function 12 = (SendIPCCall)

Function 12 sends an IPC call:

External WB 12 <ReqCode> <SendHow> “Target” <EditorHandle>
External WB 12 $$0123 $$8001 “Spectrum XCMD” ~E01234

Where: ReqCode is the Request code for this call
SendHow is how it is to be sent:
Bit 15 = 0 send to all
Bit 15 = 1 stop after one
Bits 1-0 = 00 sent to all, Target should be a null string
Bits 1-0 = 01 send to name given in Target string
Target is the name of the request procedure to send to (Set Bit 1 of SendHow).
EditorHandle holds the required DataIn block.
EditorHandle will hold the DataOut block on return.

This function allows IPC calls to be made directly from a script. Two 128 byte buffers are allocated for the DataIn and DataOut blocks. Only the first 128 bytes will be extracted from the EditorHandle for the DataIn block, and on return, the EditorHandle will always contain 128 bytes. No checks can be made that the DataOut block is not overrun, so it is up the scriptwriter not to make any IPC calls that might return more than 128 bytes in DataOut.

Function 13 = (MessageCenter)

Function 13 communicates with the MessageCenter:

External WB 13 <Action> <Type> {EditorHandle}
External WB 13 1 1 ~E01234

Where: Action = 1 add a message, EditorHandle holds input message
Action = 2 get a message, EditorHandle holds returned message
Action = 3 delete a message, EditorHandle is ignored
Type = Message ID number
EditorHandle holds the message

Refer to Apple IIgs Toolbox Reference Vol 2 page 24-14 for details on how to build the message supplied and returned in the EditorHandle.

Function 14 = (CallToolBox)

Function 13 makes ToolBox calls:

```
External WB 14  <CallNumber> {BytesToPull} {EditorHandle}
External WB 14  $$CA04  0  ~E01234
```

Where: CallNumber is the ToolBox call number for this call
BytesToPull is the number of bytes to pull from the return stack
EditorHandle holds the bytes that will be pushed onto the stack
EditorHandle will hold on return the bytes that have been pulled from the stack

Note that 'BytesToPull' is the actual byte count. As most calls are made in 16 bit mode, this value will be double the number of words that you wish to pull off the stack.

If the optional BytesToPull is not supplied, then no bytes will be pushed and none will be pulled. This allows simple Toolbox calls to be made easily. If BytesToPull is supplied, then an EditorHandle must also be supplied with the bytes that will be pushed. Up to 256 bytes may be specified for the data to be pushed onto the stack, and up to 256 bytes may be pulled off the stack.

This function allows ToolBox calls to be made directly from a script. No checks are made to see if this is a valid call, or that the correct number of bytes has been specified for the input and return stack. An attempt will be made to preserve the stack round the call, but this may not always succeed. Care should be taken to specify the call correctly.

Function 15 = (GS/OSCall)

Function 15 makes GS/OS calls:

```
External WB 15  <RefNumber> <EditorHandle>
External WB 15  $$$2012  0  ~E01234
```

Where: CallNumber is the GS/OS reference number for this call
EditorHandle holds the parameter block for this call

Up to 256 bytes may be specified for the parameter block associated with the GS/OS call.

This function allows GS/OS calls to be made directly from a script. The parameter block for that call is passed on entry, and on exit will hold any returned values.

Function 16 = (GetID)

Function 16 returns the WorkBench unique ID:

```
External WB 16  <VarName>
External WB 16  MyID
```

This function returns a unique ID to the WorkBench XCMDs. Use this ID for any calls, such as _NewHandle, that require an ID. Any Handles created using this ID will be disposed of when the script stops.

The returned value is in decimal. Where necessary, use Function 3 of the HodgePodge XCMD to convert this into a Hex value.

Function 17 = (Dereference)

Function 17 deferences a supplied Handle:

External WB 17 <Value> <VarName>
External WB 17 $$E12345 Pointer

This function dereferences a supplied value representing a Handle. Unless the Handle has been locked, it might well move in memory while a script is executing. The returned Pointer can only be truly valid for a locked Handle, so make sure that you create a locked Handle if necessary.

The returned value is in decimal. Where necessary, use Function 3 of the HodgePodge XCMD to convert this into a Hex value.

The Failed flag will be set if the supplied value does not represent a valid Handle.

Function 18 = (SaveDataFork)

Function 18 saves the EditorHandle to the data fork of a file:

EXT WB 18 ":PathName:FileName" <EditorHandle>
EXT WB 18 ":Hard.Disk:TestFile" ~E01234

This function will replace the current data fork of the file with the contents of the EditorHandle. The data fork length will be set to the length of the EditorHandle. This allows you to copy the data fork from one file to another without affecting the resource fork of of the destination file. The normal script commands will delete any existing file completely while copying.

If the file does not exist, or is already open, or any input error occurs the Failed flag will be set.

Function 19 = (LoadDataFork)

Function 19 loads the EditorHandle from the data fork of a file:

EXT WB 19 ":PathName:FileName" <EditorHandle>
EXT WB 19 ":Hard.Disk:TestFile" ~E01234

This function will load the data fork of a file into an EditorHandle. The normal script commands will not allow the loading of a ScriptEditor form a non-standard Text file.

If the file does not exist, or is already open, or any input error occurs the Failed flag will be set.

Function 20 = (getTCPIPaddress)

Function 20 returns the TCPIP Address and Port of an active connection:

EXT WB 20 <Socket> <VarName>
EXT WB 20 10 Result

Socket is the number returned from an Open TCPSocket command. The returned value is given in dotted format with comma delimiters '194,120,12,255,8,23'.

If TCP/IP is not installed, active or online, the socket is not currently open, or any input error occurs the Failed flag will be set.

Function 21 = (getMyIPaddress)

Function 21 returns the TCPIP local Address:

```
EXT WB 21 <VarName>
EXT WB 21 Result
```

The returned value is given in dotted format with comma delimiters ’194,120,12,255’.

If TCP/IP is not installed, active or online, or any input error occurs the Failed flag will be set.

Function 22 = (OpenFile)

Function 22 opens a file for reading:

```
EXT WB 22 <:Path:FileName> <VarName>
EXT WB 22 ":HardDisk:Filename" FileID
```

Opens the data fork of a file for reading, and returns the FileID. This FileID can then be used for such commands as Function 28 of the ScriptEditor.

Be careful to call Function 25 to close the file before the script stops, or you will be left with an open file which may take a reboot to close.

Be aware that the file can also be closed by a generic Close command from somewhere else in the system.

If any input error occurs the Failed flag will be set.

Function 23 = (CloseFile)

Function 23 opens a file:

```
EXT WB 23 <FileID>
EXT WB 23 2
```

Closes an open file. You should only close files that have been opened by Function 22. If you close files that have been opened by normal script commands, problems might ensue as a result. Be aware that the file might have already been closed by a generic Close command from somewhere else in the system.

If any input error occurs the Failed flag will be set.

Function 24 = (OpenListenTCPIP)

Function 24 initiates Listen requests:

```
EXT WB 24 <Port> <VarName>
EXT WB 24 12345 MasterIPID
```

The current local Address will be opened using the supplied port. The port must be given as a single word. Use the returned MasterIPID to look for incoming requests.

You must close the MasterIPID using Function 26 when you are finished, and before the script is stopped.

If TCP/IP is not installed, active or online, or any input error occurs the Failed flag will be set.

Function 25 = (ListenTCPIP)

Function 25 looks for an incoming request:

EXT WB 25 <MasterIPID> <VarName>
EXT WB 25 60 NewIPID

Use the returned NewIPID to get any data that is sent to this port.

You must close the NewIPID using Function 26 when you are finished, and before the script is stopped

If TCP/IP is not installed, active or online, or any input error occurs the Failed flag will be set.

Function 26 = (CloseTCPIP)

Function 26 closes an IPID:

EXT WB 26 <IPID>
EXT WB 26 60

An IPID that has been opened using Function 24 must be closed before the script is stopped.

Do not close IPIDs opened from the normal script TC/PIP commands.

If TCP/IP is not installed, active or online, or any input error occurs the Failed flag will be set.

Function 27 = (WakeUpScript)

Function 27 wakes up a script:

EXT WB 27 <Value>
EXT WB 27 0

Where:	0 = Stop	turn off function
	5 = OpenedDisplay	a display has just been opened
	6 = ClosingDisplay	a display is about to be closed
	8 = XFerStart	file transfer starting
	9 = XFerStop	file transfer stopped
	10 = Carrier	carrier detected/we're online/connected
	11 = NoCarrier	carrier dropped/we're offline/disconnected
	12 = DimON	Spectrum has blanked the screen
	13 = DimOFF	Spectrum has unblanked the screen
	14 = FileReceived	a file has been received successfully
	18 = MenuItemChosen	a menu item has been chosen
	21 = TCPIPConnect	a Connection has been made using TCP/IP
	22 = TCPIPDisconnect	a Disconnection has been made from TCP/IP

After the use of the Suspend Script command, this function will start up the script again when an IPC Info call with the related <value> is received.

The function will reset itself whenever it starts a script up. If you wish to use the function again, you must call the function once more. Only one value can be used at a time.

Refer to the main Spectrum Scripting and XCMD Info documentation for further details.

General Notes

This XCMD cannot easily process all the possible Tool calls, but with care and ingenuity, you can generate your own Handles by this XCMD. These can then be dereferenced, and used as result buffers. The WorkBench XCMD was not intended to be a replacement for normally coded applications, but rather a means whereby scripts, where appropriate, could have a small amount of control at system level. A control that would not otherwise be possible without a dedicated application. Used with care, the WorkBench XCMD can be a powerful adjunct to the scriptwriters repertoire.

In many of the functions, this XCMD supplies and returns raw data in a ScriptEditor Handle. It is entirely up to the script author to make sure that any data is valid data on input. Failure to make sure the data is valid could result in trashed files or applications. This obviously could have catastrophic effects on the system.

It is obviously permissable to use any method you like to process data. A ScriptEditor (TextEdit) Handle contains full 8-bit data and TextEdit will retain these values, just make sure that you preserve it in any processing you may subsequently undertake. You can use 'Make Char' to generate non-AlphaNumeric bytes when constructing data from within a script, or you can use Functions 16 and 17 of the ScriptEditor XCMD to retrieve and build data from Hex formatted strings. You may also find that Functions 3 and 4 of the HodgePodge XCMD will be useful for converting decimal values to Hexadecimal.

Input data to Functions 12-15 is passed as low data to high data bytes, left to right. Input data to Function 14 is pushed onto the stack right to left, and is pulled in the order left to right. Use SubString commands to extract bytes from the returned data as required.

Do not give any scripts using this XCMD to another user unless you are very sure that you have thoroughly tested and debugged them. You could be responsible for trashing irreplaceable files.

You may use either Decimal or Hexadecimal values where a value input is asked for. To indicate a HexaDecimal number put a '$' at the start of the number. Note that you will need to enter the '$' twice in order for it not be parsed as a Variable.

If you enter a name for ResName, then the name must match the resource name exactly or the Function will fail.

```
============================================================
PROGRAMMING INFORMATION FOR SPECTRUM EXTERNAL COMMANDS (XCMDs)
============================================================
Written by Ewen Wannop
Last updated 98-06/28 by Ewen Wannop
Spectrum copyright (c) 1989-1998 by Ewen Wannop


========
OVERVIEW
========

NOTE: Some of these calls are only sent with Spectrum 2.2 or later, and some
calls may be of use to NDAs or other applications than XCMDs.

Spectrum External Commands (XCMDs) are external modules that can monitor
Spectrum and interract with it. Spectrum XCMDs are load files structured like
an INIT file, but with the custom file/auxtype of $BC/$4080.

There are five kinds of XCMDs:

(1) Those that operate completely behind the scenes with no direct script
commands needed. For example, the Twilight II XCMD requires no script control;
it _automatically_ controls the Twilight II screen blanker based on Spectrum's
current state.

(2) Those that respond only to direct script commands. For example, the
ScriptEditor XCMD does nothing unless a script command is issued.

(3) Those that do both 1 and 2, and install Menu Items into Spectrum 2.1 or
later. For example the BinHQX Encode XCMD.

(4) Those that do 1, 2 and 3. For example, the Balloon XCMD automatically
responds to certain messages from Spectrum (e.g. to queue a file) _AND_ it
provides script commands to which the XCMD will react.

(5) A loadable Display, for example the Browser XDisplay, can also be designed
to include XCMD code that includes any of the above functionality.

Except in the case of XDisplays, when Spectrum starts up it loads an XCMD into
memory then JSLs to the XCMD's start address so the XCMD can do what's
necessary to start up. During this "initialization", an XCMD might decide it
should not be in memory and can tell Spectrum to unload this XCMD (e.g. the
TwilightII XCMD is unloaded immediately if the Twilight II CDEV is not active
and present in the system). If the XCMD returns no error to Spectrum then it is
kept in memory, and Spectrum loads the next XCMD. XDisplays must execute their
startup code when they are loaded and opened. They _must_ inactivate themselves
whenever the display is closed.

The XCMD should not perform any time-consuming processes during the
initialization routine; it should just quickly check the environment and either
install its IPC message or bail out, telling Spectrum to unload the XCMD. After
all the XCMDs have been loaded, Spectrum displays a "Please wait...starting
```

XCMDs..." message on the screen, then broadcasts an IPC message to ALL the XCMDs to give them an opportunity to perform any time-consuming startup procedures. This message will not be seen by any XDisplays.

Once an XCMD's IPC message handler is installed, various IPC messages will be received. For example, Spectrum broadcasts an "info" message to keep the XCMDs informed about certain important events occurring.

All XCMDs _must_ watch for certain events, such as the "ShuttingDown" and "srqGoAway" messages, where the XCMD _must_ clean up any handles or tools it may have started, and prepare to be shut down by the System Loader.

SPECTRUM EXTERNAL COMMANDS IN DETAIL

When the XCMD is called through its start address to initialize itself, it should quickly check for any special requirements (e.g. is a required NDA present? is System 6 in use?). If the XCMD wishes to remain active it installs an IPC message handler to watch for messages from Spectrum.

The IPC name _must_ begin with "Spectrum XCMD~" in order for the XCMD to receive messages from Spectrum. If the XCMD provides ANY script commands (via the "External" script command) then the script command name _must_ be in all capital letters after this name (because the IPC system is case-sensitive, and Spectrum converts the command name given in the "External" command to all uppercase before broadcasting the message).

For example, if you were writing the "Widget" XCMD and wanted to provide the standard "External Widget 0 {version}" command so scripts could check for the presence of your XCMD {and optionally its version number} then your IPC message _must_ begin with "Spectrum XCMD~WIDGET~".

Optionally you may follow that name with your company name and/or version number (Spectrum only cares about the START of the IPC name).

Spectrum broadcasts informative messages to all the XCMDs by using the base name "Spectrum XCMD~". When a script command is encountered, such as "External Widget 0 version", Spectrum picks up the first word after "External" (everything up to the space), converts it to uppercase, then broadcasts a message directly to the name "Spectrum XCMD~WIDGET~" (passing a pointer to the remaining data "0 version").

IMPORTANT NOTES:

+ Spectrum uses an application defined event to control the timeouts within scripts. IF YOU USE _TaskMaster YOU SHOULD NOT SET Bit13 OF THE EventMask!! This will make sure that any timeout events will not be absorbed by your XCMD, and will be seen by Spectrum on return from the XCMD.

+ An INIT/DA/CDEV could install an IPC message using the appropriate "Spectrum XCMD~" name. By watching for the "StartedUp" and "ShuttingDown" message the INIT/DA will know whether Spectrum is active or not. Remember, all extensions should also respond to the system's "srqGoAway" message!

```
+ Because XCMD names must be unique, you should register your XCMD's name with
Seven Hills Software to help avoid duplication. We also want to hear about any
XCMDs you create that might be useful for others.

As of 98-01/21 the following module names are known to be in use (there may be
more which are not documented here):
    - AlertWindow
    - Balloon
    - BatchXfer
    - BinHQX
    - Browser (XDisplay)
    - Chatterbox
    - DataBase
    - Debug
    - Figlet
    - GS+ (GSPlus)
    - HP (HodgePodge)
    - Inform
    - Kermit
    - Library
    - LineEdit
    - Lister
    - RadioCheck
    - ResEdit
    - rVersion
    - SE (ScriptEditor)
    - Speech
    - TopCat
    - TwilightII
    - WindowMgr
    - WM (WindowManager)
    - XLoader
    - Spell
    - WB (WorkBench)


=======================
SPECTRUM'S IPC MESSAGES
=======================

IPC message ranges:
     $0000-$7FFF = RESERVED for Apple's system messages
     $8000-$8800 = RESERVED for Spectrum's OUTbound messages
     $8801-$8FFF = RESERVED for Spectrum's INbound messages
     $9000-      = AVAILABLE for the external's own private use


CURRENTLY Spectrum broadcasts the following messages to XCMDs:

     $8001 = Process Script Command (to "Spectrum XCMD~CMDNAME~" stopAfterOne)
     DataIn:
       dc i2'5'          number of parameters in this block
       dc i4'Pointer to String'
       dc i2'QuoteValue'
       dc i2'TokenValue'
       dc i2'CaseSense'  $5F or $7F
       dc i2'ReplacementValue'
```

```
     DataOut:
       dc i2'recvCount'

     $8002 = Info (to "Spectrum XCMD~" toALL)
     DataIn:
       dc i2'2'          number of parameters in this block
       dc i2'infoCode'   see code table
       dc i4'PointerToSomething'
       dc i2'2'          optional data - See MenuItemChosen
    DataOut:
       dc i2'recvCount'

     $8003 = Event Info (to "Spectrum XCMD~" stopAfterOne)
     DataIn:
       dc i2'2'          number of parameters in this block
       dc i2'infoCode'   see code table
       dc i4'PointerToSomething' or dc i2'Hit value'
     DataOut:
       dc i2'recvCount'

     $8004 = Compression Request (to "Spectrum XCMD~" stopAfterOne)
     DataIn:
       dc i2'3'          number of parameters in this block
       dc i2'infoCode'   see code info
       dc i4'Handle'
       dc i2'Method'
     DataOut:
       dc i2'recvCount'
       dc i2'Method'

     $8005 = XCMD Intercommunication (to "Spectrum XCMD~CMDNAME~" stopAfterOne)
     DataIn:
       dc i2'2'          number of parameters in this block
       dc i2'infoCode'   see code info
       dc i4'Pointer to Pascal String'
     DataOut:
       dc i2'recvCount'


CURRENTLY Spectrum recognizes the following INbound messages:

     $8801 = Run This Script (send to "SHS~Spectrum~" StopAfterOne)
     DataIn:
       dc i2'1'          number of parameters in this block
       dc i4'Handle to Script'
     DataOut:
       dc i2'recvCount'

     $8802 = Insert this Menu Item(send to "SHS~Spectrum~" StopAfterOne)
     DataIn:
       dc i2'3'          number of parameters in this block
       dc i4'Pointer to Menu Item Line'
       dc i2'Item after which item is inserted' (0 to front, $FFFF at end)
       dc i2'Menu ID of menu to contain new item'
     DataOut:
       dc i2'recvCount'
       dc i2'ID of inserted Menu Item'(0 if the insertion failed)
```

```
      dc i2'ID of host Menu'(If a Hierarchical menu, this is its host menu)

    $8803 = Delete this Menu Item(send to "SHS~Spectrum~" StopAfterOne)
    DataIn:
      dc i2'3'          number of parameters in this block
      dc i2'ID of Menu item to be deleted'
      dc i2'Menu ID of menu that contains item'
      dc i2'ID of host Menu'
    DataOut:
      dc i2'recvCount'

    $8804 = Resume Script Action(send to "SHS~Spectrum~" StopAfterOne)
    DataIn:
      dc i2'1'          number of parameters in this block
      dc i2'ID of calling XCMD'
    DataOut:
      dc i2'recvCount'

    $8805 = System MenuBar Changing(send to "SHS~Spectrum~" StopAfterOne)
    DataIn:
      dc i2'2'          number of parameters in this block
      dc i2'infoCode'   see code info
      dc i2'Method'     see code info
    DataOut:
      dc i2'recvCount'
```

MESSAGE $8001 = PROCESS SCRIPT COMMAND

When a script contains a command such as:

```
        EXTERNAL CommandName <data for the XCMD>
   -or- EXT CommandName <data for the module>
```

Spectrum broadcasts a message (stopAfterOne) directly to "Spectrum XCMD~COMMANDNAME~". If your XCMD receives message $8001 you know the command is yours to process!

NOTE: If you create a "suite" of XCMDs that all use the same base name (e.g. "Ext MyCompany MyCommand") then each of your XCMDs will have to examine the data string to determine if the command really belongs to that particular XCMD. If it DOESN'T then refuse the message and the next XCMD will get a shot, otherwise process the command and accept the message.

The DataIn block holds a Pointer to a String of up to 630 characters. The actual String buffer in Spectrum is a length WORD (2 bytes) followed by 636 bytes. The original buffer might contain "Ext A 630chars", but by the time the buffer is passed to you the command info ("Ext A ") will be removed, leaving a maximum of 630 characters. {Using "External A " restricts the maximum parameter length to 636-11=625 bytes; using "External MyCommandNameHere " restricts the max parameter length to 636-27=609 bytes.}

NOTE: You MAY use any free bytes at the end of the string buffer for temporary (i.e. just during this call) storage, so long as you do not exceed the total buffer length of 2 length bytes followed by 636 data bytes.

The data that must be passed to the XCMD is entirely up to the XCMD. When the XCMD receives the string, Spectrum has already expanded the line and removed any extra spaces according to the following rules:

- Commas, Tabs, and HardSpaces (Option-Spacebar) characters that are NOT enclosed in the current Quote character are converted to Spaces.
- Two or more spaces that are NOT enclosed in the current Quote character are converted to a single Space.
- If a "#" is encountered that is NOT enclosed in the current Quote character, it and the remainder of the line is removed.

For example:

```
External YourCommand   abc    $Boot * *%%  38,48  "3 ab  *%% ^dc"  # blat
# where * is a TAB and % is an Option-Space
```

gets broadcast to "Spectrum XCMD~YOURCOMMAND~" with the following as the String parameter:

```
XXabc :Hard: 38 48 "3 ab  *%% Xc"
^^ Length WORD                ^ hex $04
```

Note that replacements have been made ($Boot, ^d), and that Commas, Tabs, HardSpaces, and extra spaces are left intact when enclosed in the current Quote character.

These rules make it easier for you to pick up a parameter using this method:

- Get a character.
- If it does NOT match the current Quote then grab bytes until you see a Space character _OR_ you reach the end of the String buffer.
- If it DOES match the current Quote character then grab bytes until you see another Quote character _OR_ you reach the end of the String buffer (and you can set a flag indicating this parameter is a string).

Anyone who installs a "Spectrum XCMD~" message handler SHOULD also provide a script command name (e.g. "Spectrum XCMD~MYCOMMAND~") and provide the following script functions:

If the XCMD has a "Preferences" dialog box, it should be presented by using the command:

```
EXT XCMDname
# note NO parameters
```

In order for scripts to know if your XCMD is present, your XCMD should recognize the following commands:

```
EXT XCMDname 0
-and- EXT XCMDname 0 <VarName>
```

When your "Spectrum XCMD~XCMDNAME~" handler receives the "process script" message, first check the Length WORD. If (Length=0) then do your preferences dialog; if you have no preferences dialog then return a "Set Failed" script.

If (Length>0) then get the next parameter. If the parameter is not "0", and you are not expecting further numbered functions, then return a "Set Failed" script (user made a syntax error requesting a function other than "0"). Otherwise, check for another parameter.

If there is NOT another parameter then just return (with no script the Failed flag will be clear). Otherwise get that parameter (the optional variable name to hold the version number).

Finally, check to see if there is still another parameter. If so return "Set Failed" (user made a syntax error by providing too many parameters). Otherwise return the script "set var VARNAME v1.0" which will set the user's VARNAME to your version number (which should be in the standard format: v1.0, v1.0b1, etc.).

MESSAGE $8002 = INFO

Various info messages are broadcast to all modules by Spectrum:

```
InfoCodes:
 1 = StartedUp       all XCMDs have been loaded
 2 = ShuttingDown    all XCMDs are about to be shut down
 3 = ScriptOn        a script has just been started
 4 = ScriptOff       a script is about to be stopped
 5 = OpenedDisplay   a display has just been opened
 6 = ClosingDisplay  a display is about to be closed
 7 = SpectrumIsIdle  called every 340 idle events
 8 = XFerStart       file transfer starting
 9 = XFerStop        file transfer stopped
10 = Carrier         carrier detected/we're online/connected
11 = NoCarrier       carrier dropped/we're offline/disconnected
12 = DimON           Spectrum has blanked the screen
13 = DimOFF          Spectrum has unblanked the screen
14 = FileReceived    a file has been received successfully
15 = DebugNormal     Debug a normal statement
16 = DebugXCMD       Debug an XCMD statement
17 = DebugSyntaxErr  Debug a script Syntax Error
18 = MenuItemChosen  A menu item has been chosen
19 = PrintJobStart   Printing has started
20 = PrintJobStop    Printing has stopped
21 = TCPIPConnect    A Connection has been made using TCP/IP
22 = TCPIPDisconnect A Disconnection has been made from TCP/IP
23 = ClockShow       The Menu bar clock is showing
24 = ClockNotShow    The menu bar clock is not showing
```

Note that calls 3-6 and 8-13 are NOT necessarily balanced--each may be called more than once before or after the corresponding opposite call. If you are interested in any of those messages you should simply keep track internally of the LAST message you received and use that as the CURRENT state.

Unless otherwise noted for a particular Info message, DataIn+4 (i4'PointerToSomething') points to a "Spectrum Parameter Block" which contains various information about Spectrum. You can actually pick up this pointer ONCE (such as when you get the "StartedUp" message) and refer to it from then on, because the parameter block is fixed in memory.

If your XCMD needs to perform any lengthy startup procedures it should wait until the "StartedUp" message has been received.

If Spectrum loads you, you MUST honor the "ShuttingDown" message to close any files, dispose of your memory handles, etc., because Spectrum will immediately

purge your code upon return from this call. {And whether or not Spectrum loads you, you should ALWAYS honor the system's srqGoAway message!}

The "ScriptOn" and "ScriptOff" messages (NOT necessarily balanced) informs you when a script is running.

The "OpenedDisplay" and "ClosingDisplay" messages (NOT necessarily balanced) informs you when an Online Display is opened or about to be closed. You might watch for these if you display windows which should be opened/closed along with a particular online display.

The "SpectrumIsIdle" message is broadcast whenever Spectrum isn't busy for a while ("idle" is considered to be "no port input while in the main SP event loop" and "no typing in the Online Display"). If a script is running, "Idle" can only occur at most "WaitFor" commands.

The "XFerStart" and "XFerStop" messages are broadcast around a file transfer session (i.e. when the file transfer dialog box appears and disappears). File transfers are most sensitive to system overhead, so during a file transfer you should be extra careful not to cause any event that might slow down (or interrupt) the system.

The "Carrier" and "NoCarrier" messages let you know when the modem is connected to another modem. While a carrier is detected you should be careful not to cause any event that might slow down (or interrupt) the system.

The "DimON" and "DimOFF" messages let you know when Spectrum itself has blanked or unblanked the screen.

The "FileReceived" message is broadcast after a file has been successfully received. For this message, DataIn+4 (i4'PointerToSomething') points to the complete (and expanded) pathname of the received file. The pointer is valid only for the duration of the Info call; if you need the path in the future you must copy it into your own internal buffers. NOTE: This message is sent in addition to the XferStarted/XferStopped messages (e.g. in a batch transfer you might see XferStarted/FileReceived/FileReceived/FileReceived/XferStopped).

The "Debug" messages are sent if the Spectrum script command "Set Debug External" has been used. "DebugNormal" is sent when the current line is from the main script; "DebugXCMD" is sent when the current line was created by an XCMD; "DebugSyntaxErr" is sent when a Syntax Error occurs.

The DataIn and DataOut blocks are redefined for the "Debug" messages, as follows:

DataIn+2	Info code
DataIn+4	Pointer to parsed line (Length word + String)
DataIn+8	Handle to script
DataIn+12	Offset in script to start of current line
DataIn+14	Offset in script to end of current line
DataIn+16	Total length of script
DataIn+16	the Spectrum error code (valid only for "DebugSyntaxErr")
DataOut+2	If zero (default), debugging continues
DataOut+2	If non-zero, no further debugging

Note that only one Debug XCMD should be active when Spectrum is running. You should make this clear in any notes that are provided with your Debug XCMD.

The "MenuItemChosen" message is sent when a menu item has been chosen but before the command is executed. DataIn+8 holds the ID of the Menu Item. If DataOut+2 is non-zero on exit, the menu command will be ignored. This allows an XCMD to take any actions necessary before the menu choice is executed, or to stop a specific menu command from being executed. An XCMD should not respond to one of its own inserted menu items when it gets this call. It should wait until it gets the "Inserted Menu Item Hit" call.

The TCP/IP 'Connected' and 'Disconnected' messages will be sent out so XCMDs will know that the normal Spectrum vectors are no longer pointing directly at the serial ports. This allows XCMDs that may rely on precise timing, such as protocol transfers, to adapt themselves to the different conditions of the Internet.

The DataIn block for the 'Connected' call will contain the following:

```
DataIn+2      Info code
DataIn+4      Address of 'SendData' vector.
DataIn+8      Address of 'ReceiveData' vector.
DataIn+12     Address of 'Flush' vector
DataIn+16     Address of parameter block
```

An XCMD can use the 'SendData' and 'ReceiveData' vectors instead of Spectrum's usual vectors. It will be marginally quicker by this route. Note that the data will be internally processed if the Telnet flag is set, so that $FF will be processed into a Telnet $FF + $FF.

To send a sequence of data, set the byte at ParmBlock+2 to '1', and then send the data byte by byte in the 'A' register. The carry flag will be clear if the data was sent successfully. Setting the transfer flag will stop the normal sending of data on a CR or when the Flush character is seen. It will then only send data at 512 byte intervals if the internal buffer fills, or when you call the Flush vector. Remember to restore this to its original value when you are finished.

Receive data normally by calling the 'ReceiveData' vector. The carry flag will be set if there was no data ready.

The parameter block holds:

```
ParmBlock+0   0=Alldata 1=Telnet 2=Getline
ParmBlock+2   0=normal processing 1=protocol transfer 'Transfer flag'
ParmBlock+4   Flush character defaults=13
ParmBlock+6   EOL characters defaults=13 and 0
ParmBlock+10  Flush frequency defaults 512 range 1-512
```

Be careful if you change any of these values yourself.

Only Spectrum v2.2 and later will send these TCP/IP related messages. For Spectrum v2.1, versions of the TopCat XCMD v1.5 and later, will send out the same messages.

The'ClockShow' and 'ClockNotShow' messages are sent out so NDAs will know that the Spectrum has taken over part of the menu bar for its own clock. The messages will only be be sent out at the change of state, so they will be matched in sequence.

MESSAGE $8003 = Event Info

Various Event Info messages are broadcast to all modules by Spectrum:

```
      1 = UnclaimedHit    Unclaimed control hit
      2 = WindowClosing   A window is about to close
      3 = Menu Hit        Inserted Menu Item Hit
```

These messages let XCMDs build windows with controls, and have Spectrum's TaskMaster loop handle events in that window. To avoid conflicts with Spectrum, controls must have IDs in the range $7001 and above (Spectrum checks the ControlID and if it's a ControlID known to Spectrum then Spectrum will act on the control, even if it's in an XCMD's window).

To avoid conflicts with other XCMDs, the XCMD must first check to make sure the _FrontWindow is their window. If so THEN they can act on the Control hit and accept the message (if the window is not theirs then ignore the hit and do NOT accept the IPC message). {For various reasons Spectrum v2.0 does not perform this kind of _FrontWindow check, which is why XCMDs must be sure to avoid controlID conflicts.}

UnclaimedHit
If there is a control hit that is unclaimed by Spectrum, the Event Info call is sent with the address of Spectrum's EventRecord in the DataIn+4 field. Accept the call only if the hit is one of your controls.

WindowClosing
If a window is about to be closed in Spectrum, the Event Info call is sent with the Window Handle in the DataIn+4 field. This call is not sent for most of Spectrum's own windows, or for Desk Accessories, but it _is_ always sent when the user presses OA-W (or clicks the window's close box) and the frontwindow is one opened by an XCMD.

If the window is recognized by an XCMD, and the call will be handled by the XCMD, accept the message and on return Spectrum will NOT close the window. Otherwise refuse the request and the next XCMD will get a shot...if no XCMD accepts the message then Spectrum will close the window itself.

Inserted Menu Item Hit
If a menu item has been inserted by an XCMD, the XCMD should monitor for this message. There will be a 'Hit Value' returned in DataIn+8 and if this corresponds to a menu item inserted by this XCMD, it should then perform any appropriate action and accept the message. DataIn+4 (i4'PointerToSomething') points to a "Spectrum Parameter Block".

MESSAGE $8004 = Compression request

Some XCMDs may may ask for data in a passed TEHandle to be compressed or uncompressed. At present only the Database XCMD will do this. One of the following messages will be sent:

```
      1 = Compress
      2 = DeCompress
```

Compress:

```
DataIn      word        3           number of entries
            word        1           info code   1=Compress, 2=DeCompress
            longword    $E01234     ScriptEditor handle
            word        0           method (0 = use best compression)

DataOut     word        x           1 = accepted
            longword    $E01234     new ScriptEditor handle
            word        x           compression method (0 = none)
```

DataIn+4 holds a TextEdit handle. DataIn+8 will normally be set to 0 (use best compression method), on return, if the message was accepted will indicate whether compression was made or not. To request a specific compression method, a specific value would be placed in DataIn+8. DataOut+2 holds the new ScriptEditor handle, the old handle is disposed of by the compression routine. DataOut+6 indicates the compression method used.

If a module can compress the data, it should do so, and return the TextEdit handle with the compressed data replacing the original data. It should then place an identifying compression method flag into DataIn+8, and accept the message.

```
DeCompress:
DataIn      word        3           number of entries
            word        2           info code   1=Compress, 2=DeCompress
            longword    $E01234     ScriptEditor handle
            word        $EE         method

DataOut     word        x           1 = accepted
            longword    $E01234     new ScriptEditor handle
            word        0           not used
```

DataIn+4 holds a TextEdit handle, DataIn+8 holds an indentifying compression method flag. If a module identifies the method and can decompress the data, it should do so, and return the TextEdit handle in DataOut+2 and dispose of the original handle. It should then accept the message.

MESSAGE $8005 = XCMD Intercommunication

The XCMD IPC structure normally has messages sent directly from Spectrum to XCMDs, with any returned message, as a result of the called XCMD Function, being sent directly back to Spectrum. Inter XCMD Communication is an extension of this normal behaviour, and allows an XCMD, or in fact any application, to talk directly to a specific XCMD, thus controlling its behaviour directly. In appropriate circumstances, this will be considerably faster than going through the normal script interface.

The WindowMgr for instance supports inbound Inter XCMD Communication, so a window could be displayed much more quickly through this method, than through a normal script command.

If there are any returned messages, they will be pointed back to the calling XCMD or application, rather than to Spectrum. The structure of the returned messages will be the same as the script command messages normally sent back to Spectrum. What these messages contain will depend on the range of responses of the target XCMD, but they will always be simply a series of script commands.

The IXC calls are sent as an IPC call to 'Spectrum XCMD~XCMDNAME~', sent with 'stop after one' and with the message number $8005. The 'XCMDNAME' is the name that you would normally use from within a script, but must be given in caps.

DataIn	word	2	number of entries
	word	0	info code 0='turn off' 1='turn on'
	longword	$123456	on entry points to the return IPC pString of the calling XCMD, and on return points to the IPC pString of the target XCMD
DataOut	word	x	1 = accepted

The 'pString' must be passed for both the 'turn on' and the 'turn off' calls. If an XCMD sends the 'turn on' command, it must follow it with a 'turn off' command as soon as it has finished communicating with the target XCMD. The two calls should bracket the session, and must be balanced.

If the 'turn on' call was accepted, the calling XCMD should make a copy of the pointer returned in DataIn+4. It must then use the pString pointed at by this pointer for all the script calls it will send to the XCMD. This will be a complete IPC call name, and it is suggested that XCMDs simply use their primary name, with a number '2' at the end, i.e. 'Spectrum XCMD~XCMDNAME2~'. However the calling XCMD must use the original IPC name when sending the 'turn off' call.

Before returning from the 'turn on' call, the target XCMD must turn on the secondary entry point, by an AcceptRequests call, for the pString name it is about to broadcast.

Now the XCMD or application can talk directly to the target XCMD and have script command responses sent directly back to itself.

Calling messages are constructed exactly as they would be put together by a script within Spectrum, except that only the portion following 'EXT XCMDName' will be passed. Only message $8001 will normally be recognised and decoded by this secondary entry point. Refer to the 'Process Script Command' $8001 for details of how to construct the calling messages, and 'Run This Script' $8801 for details of how to construct the messages that XCMDs return to Spectrum.

The calling XCMD may set its own values for QuoteCharacter etc.

If the XCMD can only handle one additional source of messages by IXC calls, it should not accept any new messages of $8005 type, if it has already got an active source connected. The active source must use its own name to turn off IXC calls. Any other XCMDs will be ignored.

Remember to call AcceptRequests to disable any secondary routines when the XCMD itself is shutdown. This should be done both from the target XCMD and the source XCMD or application. Normally this should have already been turned off by the balanced 'turn off' command from the calling XCMD.

MESSAGE $8801 = Run This Script

The XCMD may construct and send any valid script to Spectrum. Spectrum will accept the message only if it is available to process the script, otherwise it will refuse the message. If Spectrum refuses the message the XCMD has no option but to return from the call without being able to send any message to Spectrum.

If it is crucial that your message be accepted, you can remember the message and set an internal flag that can be checked whenever you receive an "Info=SpectrumIsIdle" message (at which point you can again try to broadcast the message to Spectrum).

NOTE: As long as all XCMDs follow the recommendation of formulating and returning scripts ONLY in direct response to the "Process This Script" message, all "Run This Script" messages will be accepted.

If Spectrum accepts the returned script, the script is not processed immediately; it is processed upon return from the XCMD (and at other logical times, in the case of a message being received that is not in direct response to the "Process This Script" message).

The return script can be any valid Spectrum script (from one command to an entire 64K script), with a few restrictions:

+ The script can contain Labels, but they will not be seen by any GOTO/GOSUB (or similar) command.
+ Be careful if you use Goto/Gosub/etc. command, except on the last line of your script. All these commands will pass control to a label in the MAIN script (or error if the label did not exist). If the label exists, and is found, any further lines remaining in your XCMD script will be processed before any further lines of the calling script are seen.
  Also, RETURN from a Gosub would return to the command right after the EXTERNAL command in the main script but AFTER the rest of your script has been executed first.
  {In general, you really should not return any Goto/Gosub commands.}
+ If the returned script wishes to call an XCMD, the "EXTERNAL" script command must be the _last_ command in the returned script.

The Library XCMD is an example of how long sections of script may be returned, and how the limiatations of this process work in practise.

An IPC message sent back to Spectrum is in the form of a Handle which points to ASCII text representing one or more script commands. The length of this text is the Handle length.

Spectrum will dispose of the Handle when it has finished processing the script commands. The length of this handle must be less than 64K, but it MAY cross a bank boundary.

There can be multiple statements in the script as long as each is no more than 636 chars in length. Separate each line of the script with hex #$0D (Return). Use hex #$04 where a command break (;) is required.

Note that the current Quote, Token and CaseSense characters can be obtained from the DataIn block that Spectrum sends with its message, and those current

characters must be used in your return script. TIP: Using the $Quote and $Token replacements can make things easier (e.g. SET VAR varname $QuoteThis is the string to return.$TokenM$TokenJ$TokenJ$Quote).

If you need data from Spectrum you can find most information in the Spectrum Parameter Block. For data not provided in that block you can formulate a return script that calls your XCMD again with the further data. One good use of this is in the ScriptEditor XCMD.

The ScriptEditor XCMD lets a user act on different script editors by passing EITHER an editor number OR an editor handle. For example, both of these command forms do the same thing:

```
      Ext ScriptEditor 7 3 "Sets editor 3 to this string"
 -OR- Ext ScriptEditor 7 $EditorHandle3 "Sets editor 3 to this string"
```

When the script uses $EditorHandle3, the ScriptEditor XCMD receives "~012345" as a parameter, which is the Handle to the given editor. In this case the script has no idea WHICH editor number is being referenced (and doesn't care). It just checks the handle...if it's ~000000 then it returns "Set Failed" because the editor doesn't exist, otherwise it puts the given string into that TEHandle.

When the script uses "3" (the editor number) the ScriptEditor XCMD formulates the following return script:

```
      Create ScriptEditor 3
      Ext ScriptEditor *7 $EditorHandle3
```

The first statement insures that editor number 3 is created (if there's enough memory) and the second statement causes the ScriptEditor XCMD to be called again. The "*7" is an internal/private command that informs the ScriptEditor XCMD that "here's the extra info you need to complete function 7". It picks up the handle (again verifying it's not ~000000, in case there wasn't enough memory to create the ScriptEditor) then puts the previously-supplied string into that TEHandle.

If the ScriptEditor also needed to know the current time and the name of the folder that contained Spectrum, it could formulate a script like this:

```
      Ext ScriptEditor *7 $EditorHandle3 $FullTime $Quote$Spectrumpath$Quote
```

{The parameters don't matter to Spectrum because everything after "Ext ScriptEditor" is given to the ScriptEditor XCMD for interpretation.}

MESSAGE $8802 = Insert This Menu Item

An XCMD can ask Spectrum to insert a Menu Item within a Spectrum menu.

The Menus that will allow an item to be inserted are:

File Menu	ID - 3
Edit Menu	ID - 2
Show Menu	ID - 11
Phone Menu	ID - 5
Script Menu	ID - 4
Settings Menu	ID - 8
Receive File Menu	ID - 6
Send File Menu	ID - 7
Capture to File	ID - 17
Style Menu	ID - 18
(eXtras Menu)	ID - 20
File Transfer	ID - 21

A pointer to the Menu Item Line is passed in DataIn+2. The line must be structured as a standard Menu Item line using a Hexadecimal Item ID. The line should be preceeded by a length word and followed by a zero byte:

```
dc   i'18',c'--Item String\Hxx',i1'0'
```

Spectrum will fill in the 'xx' space with the correct ID before the line is inserted. In order that Spectrum can do this, the 'xx' must be the last item on the line followed only by the zero byte. This line must remain in memory while the menu item is active. The actual menu ID that is allocated will be returned in DataOut+2. If this value is zero, then the insertion was not successful.

Once an item has been inserted, the XCMD should then monitor for any hits on these Menu IDs.

The new menu is inserted after a specific menu Item. The ID of this Item is required for the call. If this might be a newly inserted Item, rather than a standard Spectrum Menu Item, you can find the ID by first calling _CountMItems for that menu, and then calling _GetMItem repeatedly until a matched string is found.

MESSAGE $8803 = Delete This Menu Item

Whenever an XCMD is shutdown, any Menu Items which have been inserted should be deleted. It is not safe to assume that Spectrum itself is closing down, and therefore removing the menu items itself, if a shutdown call has been received. XLoader or a similar XCMD may remove or load an XCMD at any time.

The XCMD can also decide to delete menu items at any time.

MESSAGE $8804 = Resume Script Action

Will cause a script that is in ‘Suspension’ from a ‘Suspend Script’ command to resume action. If a script is performing normally, or no script is running, no further action is taken.

An example of how to use this call is when the WindowMgr XCMD receives an appropriate 'unclaimed' hit to a window control or menu that is has built. It would then need to 'wake up' Spectrum script control once more, so the correct script responses can be sent back to Spectrum.

MESSAGE $8805 = System MenuBar Changing

NOTE: This call is only recognised with Spectrum 2.2 or later

XCMDs such as the WindowMgr, have the ability to replace the Spectrum System MenuBar with their own custom MenuBar. While these custom MenuBars are active, various settings may be changed that would interfere with the proper higlighting of Spectrum's own MenuBar when it is restored. In order for Spectrum to respond correctly to its own MenuBar being replaced, the following call should be sent to Spectrum.

Some Menu equivalents within Spectrum are recognised even when the MenuBar is not showing. Use the 'Method' setting to turn off this recognition so you will be in total control of all Menu Key equivalents.

infoCode	word	0	Spectrum MenuBar has just been restored
	word	1	Spectrum MenuBar is about to be removed
Method	word	0	Allow Spectrum to see non-menu equivalents
	word	1	Spectrum will not see non-menu equivalents

External XCMDs Sample Code

```
*****************************************************************************
*
*        The Spectrum Copyright (c) 1989-94 Ewen Wannop
*
*        Sample XCMD script Module
*
*        Last updated - 14th January 94 by Ewen
*
*        Calls valid from Spectrum v2.0B35
*
*****************************************************************************
*
*        Spectrum External Script Command:
*        Filetype $BC/$4080 (Temporary auxnum)
*
*****************************************************************************
*
* Overview:
*
* Spectrum XCMDs are external modules that may be called by a script using a
* special script command 'EXTERNAL' (or 'EXT' for short).
* XCMDs allow the range of commands that a script can execute
* to be expanded without limit. The XCMDs take the form of a Load File.
*
* All XCMD modules are loaded at Spectrum boot and called from their start
* address to initialise the code within the module. By calling the start
* address, the XCMD may be coded easily in any language.
*
* When Spectrum Quits, it will broadcast a 'ShuttingDown' IPC message to all
* the modules. Each module must clean up any Handles or Tools it may have
* used and prepare to be ShutDown by the System Loader after this call.
*
* Modules may, any time its code is active, send an IPC message back to
* Spectrum. If Spectrum has no other inbound message pending, it will
* accept the message for processing later. The data is processed
* by Spectrum as though it were a script.
*
* This gives XCMD modules a way to communicate with Spectrum.
*
* If the module wishes to interrogate Spectrum, a script can be
* constructed in the form of a further 'EXT' command, thus
* allowing the module to receive answers to its questions from Spectrum.
*
*
* The XCMD in detail:
*
* When the module is called through its start address to Initialize the
* code, it will normally start up and install its IPC message handler
* using the following name:
*            'Spectrum XCMD~MyModuleName~'
* where 'MyModuleName' is a unique name for this module.
*
* When the module receives Spectrum's 'ShuttingDown' message, it disposes
```

```
* of any Handles it has not yet cleared and cleans up ready to be shut down.
*
* Note: If your code is implemented as an INIT you can watch for the "info"
* messages from Spectrum to know when Spectrum is starting up/shutting down
* all the other XCMDs. As an INIT you should also watch for the standard
* srqGoAway message.
*
*
* Spectrum's IPC messages:
*
*       $8000-$8800 = RESERVED for Spectrum's OUTbound messages
*       $8801-$8FFF = RESERVED for Spectrum's INbound messages
*       $9000-      = AVAILABLE for the external's own use
*
*
* CURRENTLY Spectrum broadcasts the following OUTbound messages:
*
*       $8001 = Process Script Command (sent StopAfterOne)
*        DataIn:
*          dc i2'4'          number of parameters in this block
*          dc i4'Pointer to String'
*          dc i2'QuoteValue'
*          dc i2'TokenValue'
*          dc i2'CaseSense'  $5F or $7F
*        DataOut:
*          dc i2'recvCount'
*
*       $8002 = Info (sent to all externals)
*        DataIn:
*          dc i2'2'          number of parameters in this block
*          dc i2'infoCode'   see code table
*          dc i4'SpectrumParameterBlock'  Pointer (see separate docs)
*        DataOut:
*          dc i2'recvCount'
*
*
* CURRENTLY Spectrum supports the following INbound messages:
*
*       $8801 = Run This Script (send StopAfterOne)
*        DataIn:
*          dc i2'1'          number of parameters in this block
*          dc i4'Handle to Script'
*        DataOut:
*          dc i2'recvCount'
*
*
* MESSAGE $8001 = Process Script Command
*
* When a script contains a command such as:
*
*         EXTERNAL TargetModuleCommandName <data for the module to process>
*    -or- EXT TargetModuleCommandName <data for the module to process>
*
* The DataIn block holds a Pointer to a GSString which may be up to 630
* characters long. The string can be any data that the XCMD and the script
* writer wishes. The string will be expanded by Spectrum, so any
* $ReplacementItems will be expanded before transmission. The actual buffer
```

```
* used by Spectrum is a 2 byte length word plus 638 bytes. You will have a
* minimum of 8 bytes free at the end of this buffer for your own use.
*
* When receiving the message $8001 the XCMD knows that is has been directed
* by Spectrum to that XCMD or to a group of XCMDs with the same base name.
* It may need to examine the data string to determine if the command is
* intended for itself or for one of the other XCMDs.
* If it ISN'T, refuse the message and the next XCMD will get a shot.
* If it IS recognized, process the command and (always) accept the message.
*
* It is highly recommended that all XCMDs respond to the script commands:
*
*         EXT TargetModuleCommandName 0 <VarName>
*   -and- EXT TargetModuleCommandName 0
*
* After accepting the message, if the optional <VarName> is present, the XCMD
* should return the version number of the XCMD in the following script:
*
*         Set Variable <VarName> <VersionNum>
*
* The Version Number should conform to the standard format eg. v1.0 or v1.0b1.
*
*
* MESSAGE $8002 = Info
*
* Various info messages are broadcast to all modules by Spectrum:
*
*        InfoCodes:
*          1 = StartedUp        all external modules have been started
*          2 = ShuttingDown     all external modules are about to be shut down
*          3 = ScriptOn         a script has just been started
*          4 = ScriptOff        a script is about to be stopped
*          5 = OpenedDisplay    a display has just been opened
*          6 = ClosingDisplay   a display is about to be closed
*          7 = SpectrumIsIdle   called every 340 idle events
*          8 = XFerStart        file transfer starting (interrupts NOT safe)
*          9 = XFerStop         file transfer stopped (interrupts ARE safe)
*          10 = IntNOTsafe      carrier detected
*          11 = IntAREsafe      carrier dropped
*          12 = DimON           Spectrum has faded the screen
*          13 = DimOFF          Spectrum has restored the screen
*          14 = FileReceived    a file has been successfully received
*          15 = Debug Normal    Line statement
*          16 = Debug XCMD      Line statement
*          17 = Debug Syntax Error Line Statement
*
*          Note that calls 3-6 and 8-13 are NOT necessarily balanced,
*          each may be called more than once without the corresponding call.
*          "Idle" is considered as "no port input while in the main SP
*          event loop". If a script is running, "Idle" can only occur at
*          most "WaitFor" commands.
*          Interrupts ARE safe at Spectrum Boot, that is after call 1.
*          You only need to monitor for call A, and turn off any
*          further interrupts until call B arrives. These calls are
*          sent immediately after the script calls 3-4, but only if
*          there is a carrier and the screen has not been faded.
*          These two calls should be used in conjunction with calls 8-9
```

```
*          to make sure that you do not cause any event that might slow
*          the system down through added interrupts if either is true.
*
*
* InfoCodes
* Normally you should not send any messages in to Spectrum during an Info
* message, but it is possible. (but _never_ send one if you get a
* "ShuttingDown" message).
*
* ShuttingDown
* If Spectrum loaded you, you MUST honor the "ShuttingDown" message
* to dispose of any of your memory Handles etc. as Spectrum will
* purge your code upon return from this call.
*
* FileReceived
* Info #14 is sent after each file has been successfully received. This call
* is sent in addition to the XFerStop call. A pointer to the pathname of
* the file is at DataIn+4. The data there will only remain valid for the
* duration of the Info call. A copy of this path should be kept if access
* to it is required later.
*
* Debug statements
* Info #15-17 are sent if the command Set Debug External has been invoked.
* Info #15 is sent when the current line is from the main script.
* Info #16 when the line was created by an XCMD.
* Info #17 is sent when a Syntax Error occurs.
*
* Parameters are passed to the IPC call by the DataIn and DataOut blocks:
*
*          DataIn+2            Info code
*          DataIn+4            Pointer to parsed line (Length word + String)
*          DataIn+8            Handle to script
*          DataIn+12           Offset in script to start of current line
*          DataIn+14           Offset in script to end of current line
*          DataIn+16           Length of script
*          DataIn+16           For Info #17 this is the Spectrum error code
*          DataOut+2           If zero (default), debugging continues if set
*          DataOut+2           If non-zero, no further debugging
*
* MESSAGE $8003 = NewInfo
*
* Various NewInfo messages are broadcast to all modules by Spectrum:
*
*          1 = UnclaimedHit    Unclaimed control in window hit
*          2 = WindowIsClosing Window about to close
*
* NewInfo Calls:
* These calls allow XCMDs to build windows with controls, and have Spectrum's
* TaskMaster loop handle events. Controls should have IDs in the range
* $6100-$67FF. Please register IDs with Seven Hills in order to avoid XCMD
* and Spectrum conflicts. These calls are sent 'Stop After One' to all
* modules, target modules should check that the hit or other content
* of the call is for them before accepting the call.
*
* UnclaimedHit
* If there are any hits in window controls that are unclaimed by Spectrum,
* then NewInfo call #1 will be sent with the address of Spectrum's EventRecord
```

```
* in the DataIn+4 field. Accept the call if the hit is one of your controls.
*
* WindowIsClosing
* If a window is about to be closed in Spectrum, NewInfo call #2 will be sent
* with the Window Handle in the DataIn+4 field. This call is not sent for
* many of Spectrum's own windows, or for Desk Accessories. However it will
* be sent for all windows opened by an XCMD that are closed by OA-W in
* Spectrum or by clicking the windows close box.
* If the window is recognised by an XCMD, and the call is to be handled within
* the XCMD, accept the call. On return, the window will not then be closed by
* Spectrum. It is then up to the XCMD to close the window when it is
* appropriate.
* XCMDs should make sure that they close all of their open windows, before
* they are finally shutdown.
*
*
* MESSAGE $8801 = Run This Script
*
* The XCMD may construct and send any valid script to Spectrum.
* Spectrum will accept the message only if it is available to process
* the script, otherwise it will refuse the message.
* If Spectrum refuses the message the XCMD has no option but to return
* from the call without being able to send any message to Spectrum.
*
* If it is crucial that your message be broadcast, you can remember
* the message and set an internal flag that can be checked whenever
* you receive an "Info=SpectrumIsIdle" message (at which point you can
* again try to broadcast the message to Spectrum).
* POSSIBLE ALTERNATIVE: A "private" XCMD could be written that could
* "queue up" these messages (i.e. XCMD broadcasts to SP; if SP refuses
* then XCMD rebroadcasts same message, but this time to the private
* XCMD that would queue it up and broadcast it in order, when SP was idle).
*
* If Spectrum accepts the returned script, the script is not processed
* immediately; it is processed upon return from the XCMD (and at other
* logical times, in the case of a message being received that is not
* in direct response to an EXT command).
*
* The return script can be any valid Spectrum script (from one
* command to an entire 64K script), with a few restrictions:
*  + the script can contain Labels, but they will not be seen
*    by any GOTO/GOSUB (or similar) command.
*  + Be careful if you use Goto/GotoNext/Gosub/GosubNext except on the
*    last line of your script. All these commands will pass control to
*    a label in the main script (or error if the label did not exist).
*    If the label exists, and is found, any further lines remaining in your
*    XCMD script will be processed before any further lines of the calling
*    script are seen.
*    Also, RETURN from a Gosub would return to the command right after the
*    EXTERNAL command in the main script but AFTER the rest of your script
*    has been executed first.
*  + if the returned script wishes to call an XCMD, the "EXTERNAL"
*    script command must be the _last_ command in the script.
*  + You must construct your script with actual CRs ($0D), where a line
*    break is required. Use the value $04 to simulate a command break (;).
*
* An IPC message sent back to Spectrum is in the form of a Handle which points
```

```
* to ASCII text representing one or more script commands. The length of this
* text is the Handle length.
*
* Spectrum will dispose of the Handle when it has finished processing the
* script commands. The length of this handle must be less than 64K, but it
* may cross a bank boundary.
*
* The text can only consist of valid script commands (see above).
* There can be multiple entries as long as each is no more than 636 chars
* in length. Separate each entry with a CR ($0D), and use $04 to represent
* the command break (;).
*
* Note that the current Quote, Token and CaseSense characters can be
* obtained from the DataIn block that Spectrum sends in its message.
*
* Spectrum uses an application defined event to control the timer within
* scripts. If you use _TaskMaster within your XCMDs, you should not set
* Bit 13 of the EventMask. This will make sure that any timeout events
* will not be absorbed by your XCMD and wil be seen by Spectrum on return
* from the XCMD.
*
*
********************************************************************************

**** Sample code in APW assembler

         absaddr  on
         65816    on
         symbol   off
         list     off
         case     on

         mcopy mod.macros

module   start

Init     anop

         phd
         phb
         phk
         plb                              set data to code bank

         sta   MyID
* Spectrum loads the ID of the module as returned from the System Loader
* on the stack before calling the module. Note this ID has been derived from
* the main ID of Spectrum. It is unique however to each module. Preferably
* use this ID for any memory you may require. In this way any Handles not
* shutdown will be disposed of by Spectrum when it Quits

         sty   LoadPath
         stx   LoadPath+2
* Spectrum loads the address of the path of the XCMD that it has just loaded
* into the X and Y registers. You may then uses this path to open resource
* forks etc.

* If you need to make any other setup calls, do them here before
```

```
* we turn on IPC requests. One very good thing you can do: Check
* for the existence of System 6 in case you were installed as an INIT
* under 5.0.4:
         pha
         _QDVersion
         pla
         and   #$0FFF
         cmp   #$0307
         bcs   InitMessages
         sec
         bra   rtlInit

* Now let the world know we are in business...
InitMessages anop
         PushLong #modName
         PushWord MyID
         PushLong #ScriptsHandler
         _AcceptRequests
         bra   rtlInit

* Spectrum pushes a zero word onto the stack just before it makes the Init
* call. It pulls this word off the stack and checks it for a non-zero value
* on return from the Init call. If this word is non-zero, the module will be
* unloaded immediately and will NOT remain in memory. If for some reason you
* do not wish your XCMD to remain in memory, simply change the first word on
* the stack after the return address and other data you may have put there,
* and your module will be unloaded. With the stack holding values as shown
* in the sample code here, the following code will change the required word.

         lda   #$8000
         sta   7,s

rtlInit  plb
         pld
         rtl


ScriptsHandler anop
* This is the actual adress that you pointed the IPC handler to
* in your _AcceptRequests call. It will be called when Spectrum
* issues an 'EXT MyModule some line here' command

OrigD    equ   1
OrigB    equ   OrigD+2
RTLAdr   equ   OrigB+1
DataOut  equ   RTLAdr+3
DataIn   equ   DataOut+4
Request  equ   DataIn+4
Result   equ   Request+2

         phd
         phb
         phk
         plb
         tsc
         tcd
```

```
         stz   <Result
         lda   <Request

TryProcessMsg anop
         cmp   #$8001                  Process Script Command message?
         bne   TryInfoMsg

         ldy   #2
         lda   [DataIn],y              read the pointer
         sta   ReadBytes+1
         iny
         iny
         lda   [DataIn],y              read the pointer
         shortm
         sta   ReadBytes+3
         longm
         iny
         iny
         lda   [DataIn],y              get the quote character
         sta   QuoteCharacter
         iny
         iny
         lda   [DataIn],y              get the token character
         sta   TokenCharacter
         iny
         iny
         lda   [DataIn],y              get the casesense value
         sta   CaseSenseFlag           $5F or $7F

         jsr   Process                 the incoming data
         bra   MessageReceived


TryInfoMsg anop
         cmp   #$8002                  Info message?
         bne   TryGoAwayMsg

         ldy   #2
         lda   [DataIn],y              get the info code
         cmp   #2                      is it the ShutDown code?
         bne   OnlyInfo

* any shut down or clean up code in here...

OnlyInfo anop
* make note/act on any other info message as desired...
         bra   MessageReceived


TryGoAwayMsg anop
         cmp   #$0003                  srqGoAway message?
         bne   exit

         lda   MyID
         ldy   #2
         sta   [DataOut],y
         iny
```

```
         iny
         lda   #$0000
         sta   [DataOut],y
* now fall through to MessageReceived

MessageReceived anop
         lda   #$8000
         sta   <Result
exit     anop
         plb
         pld
         lda   2,s
         sta   12,s
         lda   1,s
         sta   11,s
         ply
         ply
         ply
         ply
         ply
         rtl


****************************************************************
*
*        This is your internal processing code
*
*        This sample XCMD module simply takes whatever text
*        it was sent and sends it back to Spectrum quoted by the
*        text strings shown at the end of the Data section.
*        Your XCMD can do whatever it likes...
*
****************************************************************

**** start of sample code

Process  anop

* If you have constructed a set of XCMDs with the same base name you
* may need to examine the data at this point first to determine if the
* message is really for you.
*
* You can do what you like here. Remember that you do not know what
* display Spectrum is showing, if any, unless you check in the
* Spectrum data block. Normally if you need to find out information
* from Spectrum, you would send it a return message with a line of
* script in this form:
*
*        EXT ModuleName $EditorHandle0 $FileXferPath etc.
*
* On return from your XCMD, Spectrum will expand this line
* send it back to you (as another $8001 message).
* Of course you could send a message to another module, or
* just accept the message and do nothing, or chat away to Spectrum
* all day...
*
* When Spectrum sends a ModSaysMessageInfo call, you are given a
```

```
* Pointer to the Parameter block within Spectrum. Please refer to
* the separate notes for full details of this block and how to use it.
*
* Note that $EditorHandle0 substitution returns ~000000 if that
* ScriptEditor is not in use. Otherwise it will be the Handle
* of the ScriptEditor record. You may work with this Handle,
* provided you do not destroy the validity of the TERecord it contains.
* If you are not sure whether an $EditorHandle0 is in use or not,
* you can issue the 'Create ScriptEditor <num>' command which will
* initialise an editor if it does not yet exist.
*
* ScriptEditor Handles are actually TextEdit Handles and not ordinary
* Handles. You must respect their structure and only use TE commands
* with them. If you handle the data directly, you must make sure that
* the style reference, if it exists, continues to be a valid structure.
*
* Note that if you need Zero page space, you may create this space on
* the Stack that called you. Spectrum has a 4K stack available.
* When you are called the first page of the current zero page is in use
* by Spectrum and must be preserved.
*
* If you create any handles using a unique ID, make sure they are all
* disposed of when your module receives the 'shuttingdown' call.
*

        ldx   #0
        jsr   ReadBytes
        sta   Size+2                    of string being sent
        clc
        adc   #PostTitleEnd-PostTitle
        adc   #PostTrailEnd-PostTrail
        sta   Size

        pha
        pha
        pea   0
        PushWord Size
        PushWord MyID
        PushWord #$0008
        PushLong #0
        _NewHandle                      get some memory
        PullLong ModDataIn+2

        phd                             preserve zero page
        pha                             make some space
        pha                             on stack for next code
        pha
        pha

        tsc
        tcd                             new zero page

        lda   ModDataIn+2
        sta   0
        lda   ModDataIn+4
        sta   2
        ldy   #0
```

```
lda   [0],y
sta   4
iny
iny
lda   [0],y
sta   6

PushLong #PostTitle
PushLong 4
PushLong #PostTitleEnd-PostTitle
_BlockMove

clc
lda   4
adc   #PostTitleEnd-PostTitle
sta   4
lda   6
adc   #0
sta   6

clc
lda   ReadBytes+1
adc   #2
adc   Counter
tax
lda   ReadBytes+3
and   #$ff
adc   #0
pha
phx
PushLong 4
pea   0
sec
lda   Size+2
sbc   Counter
sta   Size+2
pha
_BlockMove

clc
lda   4
adc   Size+2
sta   4
lda   6
adc   #0
sta   6

PushLong #PostTrail
PushLong 4
PushLong #PostTrailEnd-PostTrail
_BlockMove

pla
pla
pla
pla
pld                              restore zero page
```

```
         pea   $8801                    modSaysMessageSent
         pea   $8001                    stop after one
         PushLong #extString
         PushLong #ModDataIn
         PushLong #ModDataOut
         _SendRequest

         clc
         rts


ReadBytes lda  >0,x
         rts


**** end of sample code


*****************************************************************
*
*        This is your data block
*
*****************************************************************

ModDataIn dc   i2'1'
          dc   i4'0'
ModDataOut ds  4


* Some flags, fields, etc.

MyID     ds    2
Counter  ds    2
Size     ds    4
LoadPath ds    4

* These will be the current values used by Spectrum

QuoteCharacter ds 2
TokenCharacter ds 2
CaseSenseFlag  ds 2


**** Message names

modName  str   'Spectrum XCMD~YOURMODULENAME~'
* This is the name the System will know you by. It should be
* unique, so replace 'YOURMODULENAME' with a name of your choosing.
* As of 94-09/24 the following module names are known to be in use
* (there may be many more which are not documented here):
*    CoolCursor - controls CoolCursor
*    Speech - speaks using ByteWorks' Talking Tools
*    TwilightII - controls TwilightII
*
* Spectrum broadcasts all INFO messages to the root name
* 'Spectrum XCMD~'.
```

```
*
* For the Process Script Command message Spectrum constructs the
* broadcast name from the data in the EXT script command. Because the
* IPC system is case senstivie, Spectrum converts the module name into
* uppercase letters. For example, if a script contain this command:
*     EXT SampleCommand Sample data string
* Spectrum broadcasts the message to:
*     Spectrum XCMD~SAMPLECOMMAND~
*
* You could build a suite of commands by using a split name:
*     EXT YourCompany~SampleCommand Sample data string
* A script can send a message to all your XCMDs this way:
*     EXT YourCompany Sample data for all XCMDs
* Or to a specific XCMD this way:
*     EXT YourCompany~SomeCommand

extString str  'SHS~Spectrum~'
* Name to use when sending IPC messages to Spectrum

PostTitle anop
         dc    c'Display "This text was received from Spectrum '
         dc    c'by an XCMD Module - ',i1'39'
PostTitleEnd anop

PostTrail anop
         dc    i1'39',c' and was returned by the module to Spectrum..."'
PostTrailEnd anop

         end
```

```
This document describes how to use the Spectrum port driver (in colour :-)
Last update: 25th August 1994 for v2.0d2
See bottom for list of changes

If you are unfamiliar with Merlin:

ADRL - defines a long word (ADdRess Long)
DA   - defines a word (Define Address)
DS   - defines a number of bytes, initialised to 0 (Define Storage).
       APW/MPW use this
STR  - defines a pstring

*-------------------------------------------------

HEADER          STR    'Spectrum',AA,' port driver v2.0'

DRVRBLOCK   EQU  *
INITDRVR    ADRL 0      ;Initialise the driver
SHUTDOWN    ADRL 0      ;Shutdown the driver
VARIABLES   ADRL 0      ;Address of Variables
INITPORT    ADRL 0      ;Initialise to port data block and starts buffering
KILLPORT    ADRL 0      ;Stops buffering characters (doesn't change setup)
CHAROUT     ADRL 0      ;Send a character
DROPDTR     ADRL 0      ;Drops DTR (A=VBL count before raised again)
SENDBREAK   ADRL 0      ;Sends a break
STATUS      ADRL 0      ;Retrieve status word
CLEARFLAGS  ADRL 0      ;Clears XON/XOFF stuff
POLLDRIVER  ADRL 0      ;reserved / unused
SETRTS      ADRL 0      ;Manual RTS control
GETCHAR     ADRL 0      ;Gets the next character from the buffer
POSTCHAR    ADRL 0      ;Posts a character into the buffer
            ADRL 0

*-------------------------------------------------

VARIABLES   EQU *

BUFFBASE    ADRL 0      ;Pointer to start of buffer
BUFFSIZE    ADRL 0      ;Size of buffer in bytes (max = 64K)
BUFFPTRI    DA 0 ;Next free location in buffer
BUFFPTRO    DA 0 ;Next character from buffer
PORT        DA 0 ;Port definition (set by caller)

DATABLK                      ;Data block in same format as BRAM
PMFLAG      DS 1
LINELEN     DS 1
DELLF       DS 1
ADDLF       DS 1
ECHO        DS 1
BUFFERING   DS 1
BAUD        DS 1 ;Used
DATASTOP    DS 1 ;Used
PARITY      DS 1 ;Used
```

```
DCD         DS 1
DSR         DS 1 ;Used
XONXOFF     DS 1 ;Used

INFLOWFLAG  DA 0 ;0=all ok, 1=reached 10%, 2=reached 5%

HSPOLLCOUNT DA 0 ;Current handshake flag check count down
HSPOLLSTART DA 0 ;Start value for handshake flag check

USERID      DA 0 ;my user ID

VERSION     DA 0 ;driver version = $0200
*-------------------------------------------------
```

All routines are to be called in native mode with 16 bit acc and index registers. On exit, the registers will be:

A = ? (unknown)
X = ? (unknown)
Y = ? (unknown)
DBK = unchanged
P = ? - carry will indicate any errors (see below), registers will be 16 bit m/x
S = unchanged

All routines are to be JSLed.

INITDRVR - Initialises the driver when loaded. This call is made once only and does not touch any ports. There are no input parms. If the carry is set on exit, the accumulator will contain an error code. INITDRVR also makes a call to CLEARFLAGS, and installs a custom IPC request procedure.

SHUTDOWN - Shuts down the driver prior to it being disposed off. This call is made once only, and does not touch the ports. It simply turns of interrupts, and unhooks itself from the ROM. If you wish to initialise the ports before shuttting down, then call INITPORT first. There are no input parms. If the carry is set on exit, the accumulator will contain an error code. This routine may leave the port in an unknown state unless INITPORT has been called. SHUTDOWN also removes the IPC request procedure.

INITPORT - Uses the PORT word to decide which port to use, and the DATABLK to initialise the port correctly. Characters are from then on buffered into the data buffer, using BUFFPTR, BUFFBASE, BUFFSIZE, BUFFPTRI and BUFFPTRO. This routine does not drop DTR. As mentioned, the inputs are PORT, DATABLK and the BUFF values. If the carry is set, INITPORT was unable to find/use the port, and an error code will be in accumulator.

The DSR/DTR handshake setting in the IIgs Control Panel decides whether to use hardware handshaking.

The following values are used from the DATABLK ({} indicates NOT in SSC mode, [] indicates ONLY in SSC mode):

BAUD: 1=50, 2=75, 3=110, 4=134.5, 5=150, 6=300, 7=600, 8=1200, 9=1800,
10=2400, 11=3600, 12=4800, 13=7200, 14=9600, 15=19200 {, 16=38400,
17=57600 }

DATASTOP: 0=5/1, 1=5/2, 2=6/1, 3=6/2, 4=7/1, 5=7/2, 6=8/1, 7=8/2

PARITY: 0=Odd, 1=Even, 2=None [, 3=MARK, 4=SPACE]

Error codes:

0001 = AppleTalk is using that port.
0002 = There is no Super Serial Card in that slot.
0003 = The slot is not switched to the appropriate setting.

KILLPORT - Simply turns off buffering for the port.

CHAROUT - Send the character in the accumulator to the port AS IS. On exit, the carry will be clear if the character was sent OK. If the character is unable to be sent after 1 second of trying, the routine will return with the carry set, and the character NOT sent.

DROPDTR - Drops DTR for the port. The X register should contain the number of VBLs to wait before raising it again. On smart modems, this value will depend on the S register setting for DTR loss detection. Normally a second or two will suffice. A VBL count of 0, indicates that DTR should not be raised again.

SENDBREAK - Sends a break, with the X register indicating how many VBLs to send the break for. If X = 0, then the break is left on.

STATUS - Gets the status of the port.

For the Z8530, the following bits are used, and returned in A.

0-2	unused (zero)
3	GPi line status (normally DSR{really DCD for smart modems})
4	unused (zero)
5	HSKi line status (normally CTS)
6	Tx Underrun error
7	Break/Abort detect
8-15	unused (zero)

You should only need bits 3 and 5, however I have included the others for compatability with the serial port firmware. I have not included the Rx and Tx bits which the firmware returns. NOTE: These bits reflect the status of the relevant pins, and not the state mentioned in "normally...".

For the SSC, the following bits are used, and returned in A.

0	parity error (if set)
1	framing error (if set)
2	overrun error (if set)

```
4           unused (zero)
5           DCD (0 = true)
6           DSR (0 = true )
7           unused (zero)
8           CTS (0 = true)
```

CLEARFLAGS - Clears the following internal flags.

none

POLLDRIVER - reserved / unused

SETRTS - Allows manual control of the RTS line. A contains a boolean value. 0000=reset so that flow continues. <>0 = set it so that flow stops.

GETCHAR - Returns the next character in the buffer. Carry set if non available.

GETCHAR always checks to see if there is a pending send, and sends that first.

The variable HSPOLLCOUNT (HandShake Poll Counter) is decremented each time through GETCHAR. When it reaches zero, handshake processing occurs, and the count is reset to the value in HSPOLLSTART. HSPOLLSTART defaults to 1024, but can be changed by the application.

POSTCHAR - Posts the accumulator (low 8 bits) into the buffer as if it came in off the port.

XON/XOFF STUFF

If the XONXOFF flag is set, XON/XOFF flow control is enabled.

If the buffer free space drops to 10% of the total size, an XOFF will be sent out, using CHAROUT. If free space drops to 5%, another XOFF is sent through CHAROUT. It is up to the application to make sure that POLLDRIVER is called so that it can clear th 20% when necessary. The application can test INFLOWFLAG to see if it has been cleared correctly.

CTS/RTS STUFF

If the DCD flag is set, CTS/RTS flow control is enabled.

If the buffer drops to three characters free space left in the buffer, RTS stops the flow. GETCHAR/POLLDRIVER then re-starts it once free space has gone about 5%.

Data is only sent through CHAROUT when CTS indicates it is OK.

CHAROUT LOGIC

CHAROUT checks the interrupt flag (65C816 processor status register) on entry, and if set will only perform one attempt at sending the character. If the flag is clear, CHAROUT tries as many times as possible in 1 second. It is up to the caller to test the results of CHAROUT and re-queue the send if required. (The main interrupt handler in the driver does this, and sets a flag indicating there is a send pending (eg. if it is sending an XOFF). Both GETCHAR/POLLDRIVER and the main interrupt handler are capable of re-trying the send each time they are called.)

IPC REQUEST PROCEDURE

The driver also installs an IPC request procedure, for utilities and other routines to extract information from the port driver.

Your initial request should be sent as sendToName+stopAfterOne to “Seven Hills~Spectrum port driver~”. Subsequent calls can be made using the returned user ID.

dataIn is ignored, dataOut points to the following response buffer:

```
recvCount   Word        System reserved word
blockAddr   LongWord    Pointer to DRVRBLOCK
drvrVersion Word        The integer value $0200
userID      Word        The driver’s user ID
```

END

Spectrum Parameter Block

Updated: Oct 1998

NOTE: There are additional parameters described in this document which are specific to Spectrum v2.2 and later. Check the Word at VariableBlock-$02 to see which version of Spectrum is currently running:

$01	=	Spectrum v1.0
$02	=	Spectrum v2.0
$03	=	Spectrum v2.1
$04	=	Spectrum v2.2

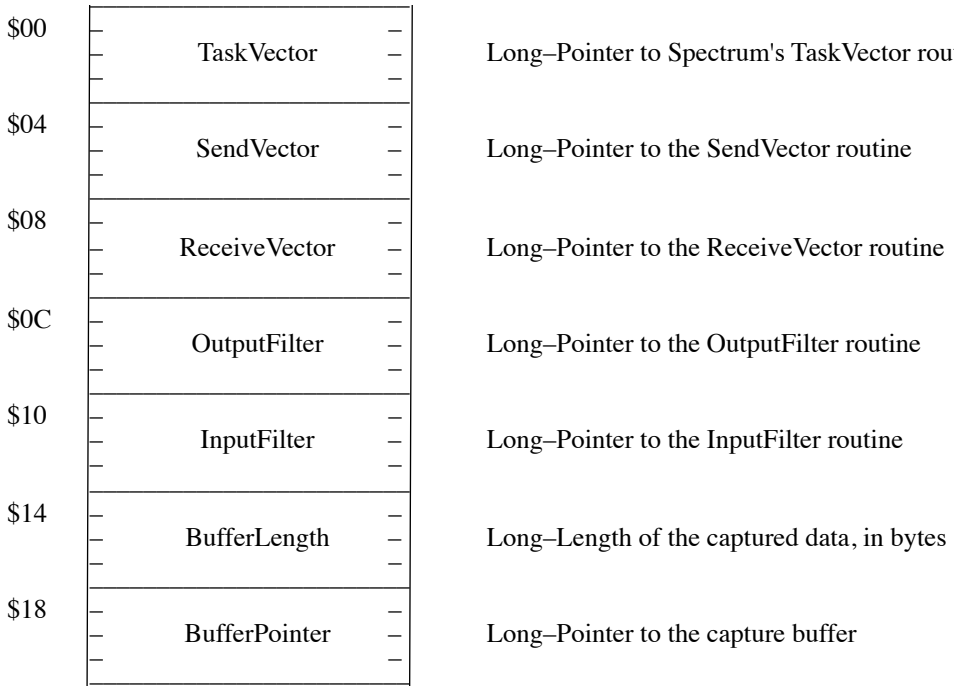
When a display is first called through its `Init_Driver` call it must set various pointers to Spectrum routines. It finds the pointers in the following parameter block within Spectrum. This block also holds other useful vectors and addresses. The pointer to this block is passed on the stack (See `Init_Driver` call).

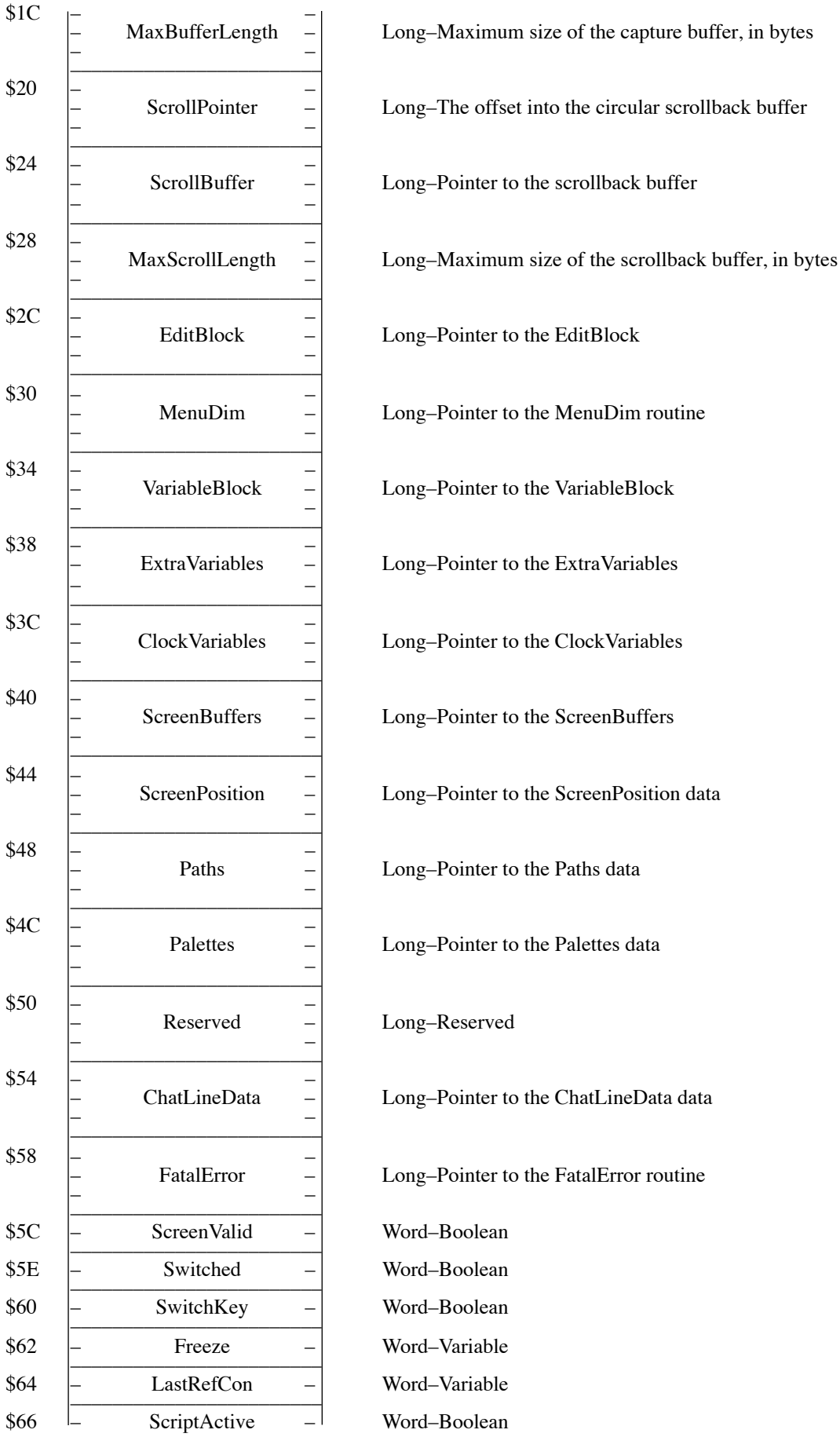
Some of these pointers may also be of interest to those writing XCMDs. Many of the vectors can be called directly from an XCMD, and many of the data blocks and flags may be read directly for use within the XCMD. A pointer to this block is passed when an XCMD Info call is made.

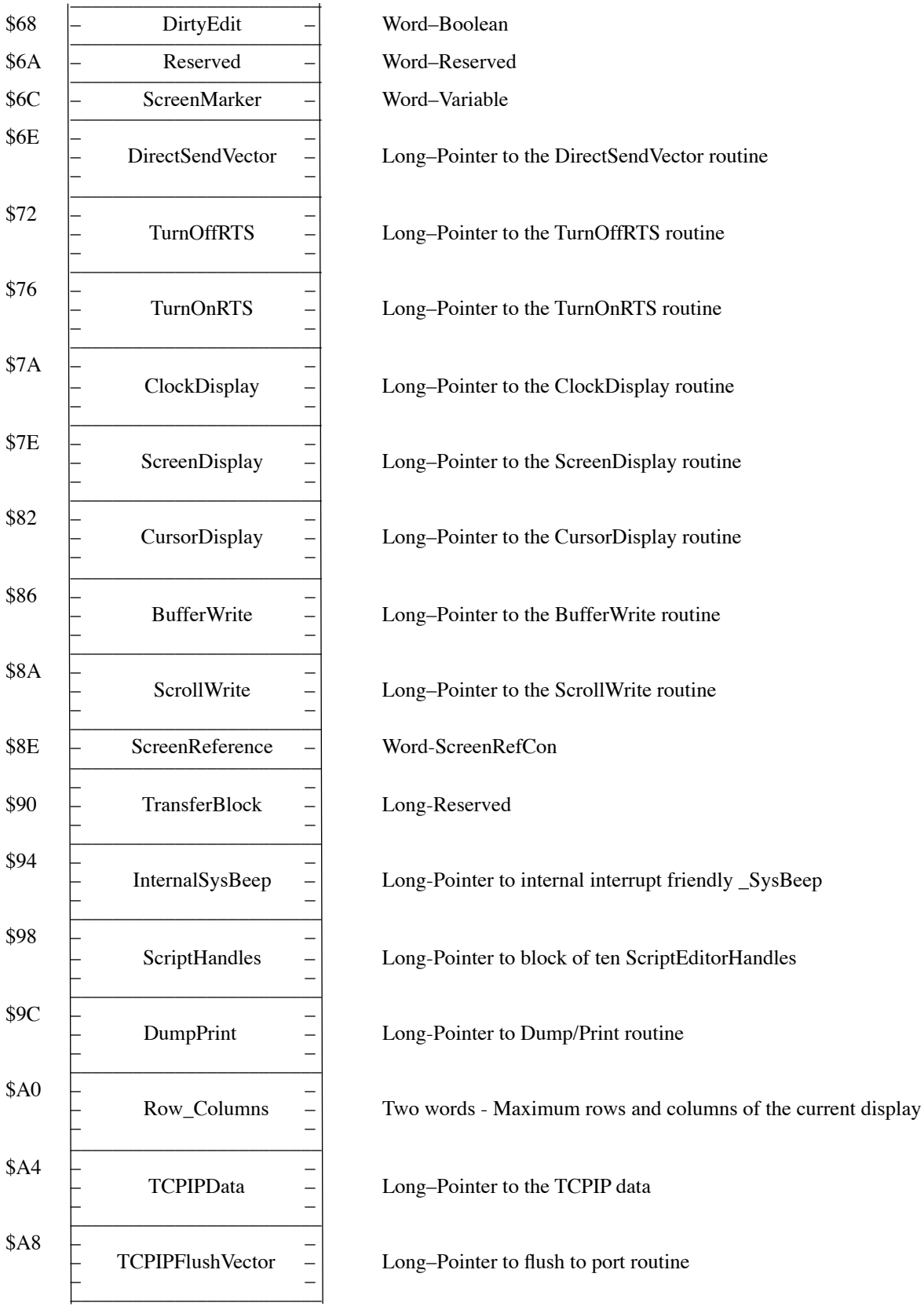
Along with the Driver Parameter Block, the Spectrum Parameter Block forms the main interface between Spectrum and its display drivers. Most of this block contains pointers to various calls, some of which must be supported within the display driver. The rest of the block contains pointers to further data blocks or contains actual data which may be read directly.

Please note that pointers only hold the address of their related routine. You cannot make a JSL to any of the entries within the Spectrum parameter Block. You must use the data in these pointers to build JSL calls that you will use to make the actual target call. Please check with the appropriate entries for further details.

All calls are made in full native mode and must be entered by a JSL.




$1C
MaxBufferLength
Long–Maximum size of the capture buffer, in bytes
$20
ScrollPointer
Long–The offset into the circular scrollback buffer
$24
ScrollBuffer
Long–Pointer to the scrollback buffer
$28
MaxScrollLength
Long–Maximum size of the scrollback buffer, in bytes
$2C
EditBlock
Long–Pointer to the EditBlock
$30
MenuDim
Long–Pointer to the MenuDim routine
$34
VariableBlock
Long–Pointer to the VariableBlock
$38
ExtraVariables
Long–Pointer to the ExtraVariables
$3C
ClockVariables
Long–Pointer to the ClockVariables
$40
ScreenBuffers
Long–Pointer to the ScreenBuffers
$44
ScreenPosition
Long–Pointer to the ScreenPosition data
$48
Paths
Long–Pointer to the Paths data
$4C
Palettes
Long–Pointer to the Palettes data
$50
Reserved
Long–Reserved
$54
ChatLineData
Long–Pointer to the ChatLineData data
$58
FatalError
Long–Pointer to the FatalError routine
$5C
ScreenValid
Word–Boolean
$5E
Switched
Word–Boolean
$60
SwitchKey
Word–Boolean
$62
Freeze
Word–Variable
$64
LastRefCon
Word–Variable
$66
ScriptActive
Word–Boolean


$68
DirtyEdit
Word–Boolean
$6A
Reserved
Word–Reserved
$6C
ScreenMarker
Word–Variable
$6E
DirectSendVector
Long–Pointer to the DirectSendVector routine
$72
TurnOffRTS
Long–Pointer to the TurnOffRTS routine
$76
TurnOnRTS
Long–Pointer to the TurnOnRTS routine
$7A
ClockDisplay
Long–Pointer to the ClockDisplay routine
$7E
ScreenDisplay
Long–Pointer to the ScreenDisplay routine
$82
CursorDisplay
Long–Pointer to the CursorDisplay routine
$86
BufferWrite
Long–Pointer to the BufferWrite routine
$8A
ScrollWrite
Long–Pointer to the ScrollWrite routine
$8E
ScreenReference
Word-ScreenRefCon
$90
TransferBlock
Long-Reserved
$94
InternalSysBeep
Long-Pointer to internal interrupt friendly _SysBeep
$98
ScriptHandles
Long-Pointer to block of ten ScriptEditorHandles
$9C
DumpPrint
Long-Pointer to Dump/Print routine
$A0
Row_Columns
Two words - Maximum rows and columns of the current display
$A4
TCPIPData
Long–Pointer to the TCPIP data
$A8
TCPIPFlushVector
Long–Pointer to flush to port routine

TaskVector

If you plan to manage your own **TaskMaster** loop, and wish to call up the event handling within Spectrum yourself, this vector gives you a pointer to a function within Spectrum that will call Spectrum's main **TaskTable**.

*WARNING: You will be calling the main routines within Spectrum, but you will **not** be passing through all of the code that Spectrum normally uses. You should use this call with **extreme caution** with a full understanding of how TaskMaster would normally be handled in a program loop. Failure to observe these conditions can cause unpredictable results.*

Example:

```
YourEventLoop   anop
                pha
                PushWord EventMask      Required mask
                PushLong #EventRecord   Pointer to your EventRecord
                _TaskMaster
                pla
                asl    a                multiply to produce two byte offset
                tax                     transfer to X register
                jsl    >TaskVector      Calls Spectrum TaskTable
```

SendVector

A pointer to the keyboard serial port output call. If **Half Duplex** is active a copy of the outgoing byte will be placed into the inputstream so it also shows on screen. *NOTE: The **DirectSendVector** does not place a copy of the outgoing byte in the input buffer. Use this alternative vector where it is inappropriate to respect the **Half Duplex** flag.*

Example:

```
                lda    Data             Data to output
                jsl    >SendVector
```

On return from the **SendVector** call the carry flag will be clear if the data was successfully transmitted. If the data was not transmitted successfully, an internal routine within Spectrum will have attempted to handle the problem by repeatedly trying to transmit the data. If this still fails to transmit the data, the continuous “try to transmit” loop will appear to have caused a program hang. If the user then aborts this loop by pressing OA-Period or ESCape the **SendVector** call will return with the carry flag set.

ReceiveVector

A pointer to the main serial port input call. No conversion or other processing is made by this call.

Example:

```
GetData anop
                jsl    >ReceiveVector   Input data
                bcc    GetData
                sta    ReceivedData     Data in A register
```

On return from the **ReceiveVector** call the carry flag will be clear if data was pending and has been returned in the A register. If no data was pending, the carry flag will be set. You would normally then loop round the same call till data was received signalled by the carry flag being clear.

OutputFilter

A pointer to a call that will process the data in the A register with any **Keyboard Filter Table** that is currently active. If no Keyboard filter cable is active, this call will return the A register unchanged.

Example:

```
lda    Data
jsl    >OutputFilter
sta    ProcessedData
```

Note that the settings in the **Online Display Settings** dialog are not applied to the **OutputFilter**. They are only applied to the **InputFilter**.

InputFilter

A pointer to a call that will process the data in the A register with the **Convert to low ASCII** setting from the **Online Display Settings** dialog and any **Display Filter Table** that is currently active.

Example:

```
lda    Data
jsl    InputFilter
sta    ProcessedData
```

On return from the call the carry flag will be clear. However if the script command **Set SendLFs** is currently set at **ON**, any line feeds that are encountered will result in the call returning with the carry flag set. It is up to your display to either keep the linefeeds or dispose of them as it wishes. If a **Display Filter Table** is currently active and any translation results in a Null character, the call will also return with the carry flag set.

BufferLength

This holds the actual length of capture data within the **Online Buffer** in a LongWord.

Note that this is an actual value and is not a pointer. Do not alter this value under any circumstances.

BufferPointer

A pointer to the start of the **Online Buffer**. Note that this value may change at any time through user interaction with the capture buffer size.

MaxBufferLength

This holds the actual size of the **Online Buffer** in a LongWord.

Note that this is an actual value and is not a pointer. Do not alter this value under any circumstances.

ScrollPointer

This holds an offset into the circular **ScrollBack Buffer**. If Bit 15 is clear, the pointer will actually point to the first byte after the end of valid data. If Bit 15 is set, the buffer will have wrapped round upon itself. The pointer in this case will point to the first byte after the end of the valid data within the circular buffer, this will also be the first byte of valid data. *Note that this is an actual value and is not a pointer. Do not alter this value under any circumstances*.

ScrollBuffer

A pointer to the start of the **ScrollBack Buffer**. Note that this value may change at any time through user interaction with the scrollback buffer size.

MaxScrollLength

This holds the actual size of the **ScrollBack Buffer** in a LongWord.

Note that this is an actual value and is not a pointer. Do not alter this value under any circumstances.

EditBlock

A pointer to this block of data for use with **Editor** records:

```
EditBlock        anop
RecordHandle     ds     4               Handle holding Edit data
StyleHandle      ds     4               Handle holding Style data
StartSelection   ds     4               Offset for start of highlighted text
EndSelection     ds     4               Offset for end of highlighted text
                 ds     2               Reserved
EditFileType     ds     2               Boolean flag: Text = false, TeachType=true
```

Whenever an online display's Editor has control, Spectrum's own editor window will be closed and any existing **TextEdit** record will have its contents transferred to memory handles (and the handle pointers stored into the **EditBlock**).

If you support a custom editor you are free to use these handles as you wish . You must however respect that these are memory handles and be careful what you do with them. If you dispose of the Handles you must also zero the pointers so Spectrum knows the Handles have been disposed of and does not attempt to use them.

If you modify the contents the **RecordHandle**, you should also modify the **StyleHandle**, or dispose of the **StyleHandle** and zero its pointers. The **StartSelection** and **EndSelection** offsets should also be changed to reflect any changes in the **RecordHandle**. They can be set to zero if you do not use a selection in your Editor.

If there are no current handles in the **EditBlock** it is permissible to create new ones. If you have no **StyleData** which conforms to **TEFormat** structure, simply leave the block with the **StyleHandle** pointer zeroed out.

The **EditBlock** allows the interchange of Editor data between different types of screen editors. If you have transferred a **TeachType** record to the **RecordHandle** pointers, you should set the **EditFileType** flag true. For a simple block of text, or a block without a **StyleHandle**, you should set this flag false.

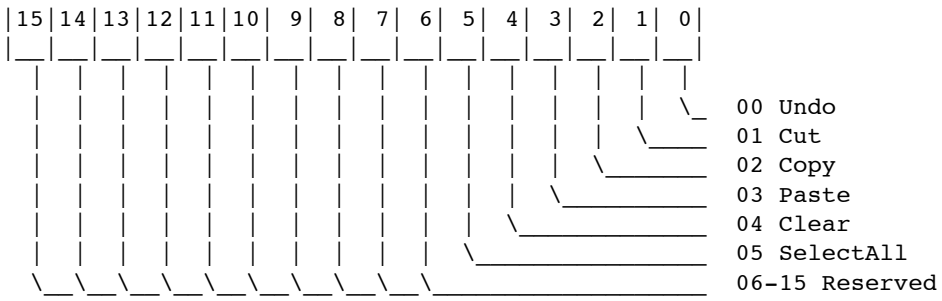
MenuDim

A pointer to a call that will Enable and Disable the Edit Items in the Edit Menu. This allows displays using the SHR desktop to control the Edit Menu Items directly. Spectrum will restore the standard Menu Item settings when your display is closed.

Example:

```
                 lda    Mask
                 jsl    MenuDim
```

Setting a bit enables the menu item; clearing a bit disables the menu item.



VariableBlock

Pointer to the main block of settings that get saved to disk in Spectrum's preference file.

You may only directly alter those variables that are marked with a "Yes" as being write enabled. Be careful in changing even those values, as you may have to take some other action as well as just changing the value of the flag. **If in doubt, do not alter any of these values!**

The **Default** and **Range** entries show the default state of the flag. Where an entry has been made for **Range**, the range is also shown. A **T/F** entry for **Range** shows that there are only two states for the flag, **True** (non-zero) or **False** (zero).

Offset	Type	Write	Default	Range	Description
-$02	Word	No	$03	$00-$03	Spectrum release number, Spectrum v2.1 = $03
+$00	Long	No	-	-	Length of Parameter block, varies depending on version
+$04	Word	No	$01	$01-$12	Baud rate, $01=Default, $02-$12 = 50-57600 baud
+$06	Word	No	$01	$01-$18	Word format, 8N1 - 5E2
+$08	Word	No	True	T/F	Xon/Xoff handshake, True = ON
+$0A	Word	Yes	False	T/F	Duplex, True = Half duplex
+$0C	Word	Yes	False	T/F	Echo to port, True = Echo ON
+$0E	Word	Yes	False	T/F	Linefeeds added to output, True = LineFeeds ON
+$10	Word	No	$8002		Serial slot. Bits 0-2 = actual slot, Bit 15 is set for internal ports
+$14	Word	Yes	$00		The LineDelay on CR output in 16/60th seconds
+$16	Word	Yes	$00		The CharDelay on character output in 1/60th seconds
+$18	Word	Yes	False	T/F	Show control characters, True = ON
+$1A	Word	Yes	False	T/F	Key click, True = ON
+$1C	Word	Yes	True	T/F	Delete/backspace key translation, True = ON
+$1E	Word	No	True	T/F	Online buffer/file capture flag, True = ON
+$24	Word	Yes	False	T/F	Autoreceive B+, True = ON
+$26	Word	Yes	True	T/F	Send ahead for B+ and Zmodem, True = ON
+$2C	Long	No	$080088EA		The current font, Default = Spectrum 8
+$32	Word	Yes	True	T/F	Auto resume of B+ , True = ON
+$34	Word	Yes	True	T/F	Binary II up, True = ON
+$36	Word	Yes	False	T/F	Binary II down, True = ON
+$38	Word	Yes	$7F	$7F-$FF	Convert to low ASCII, $7F = ON
+$3A	Word	Yes	True	T/F	CA keys active, True = ON
+$3C	Word	Yes	False	T/F	Auto buffer control, True = ON
+$3E	Word	Yes	False	T/F	Use custom editor, True = ON

+$42	Word	Yes	True	T/F	Strip extra line feeds to display, True = OFF
+$46	Word	Yes	True	T/F	Pad empty lines on text send, True = ON
+$48	Word	No	$3E	$00-$FF	The prompt value used by text send
+$4A	Word	Yes	False	T/F	ProDOS Xmodem, True = ON
+$56	Word	Yes	False	T/F	Clock display, True = ON
+$58	Word	Yes	True	T/F	Autosave settings at Quit, True = ON
+$5C	Word	Yes	True	T/F	Show text and save to buffer on send, True = ON
+$5E	Word	Yes	True	T/F	Status line, True = ON
+$62	Word	Yes	True	T/F	Sounds switch, True = ON
+$64	Word	Yes	$14	$00-$FF	Timeout on prompted text send
+$66	Word	Yes	$FFFF	$0000-$FFFF	Number of Zmodem errors allowed
+$80	String	Yes	ATZ√ATE0V1Q0S7=45		Modem Init string, 1 length byte plus 40 characters
+$AA	Word	No	$0A	$00-$FF	Wait between dials
+$AC	Word	No	$0A	$00-$FF	Number of redials
+$AE	Word	No	$3C	$00-$FF	Wait for answer
+$E0	Word	No	False	T/F	Chatline active, True = ON
+$E2	Word	Yes	True	T/F	Auto-adjust duplex while chatline active, True = ON
+$E6	Word	Yes	False	T/F	Send text by prompt, True = ON
+$E8	Word	Yes	True	T/F	Scrollback, Raw = OFF, Filtered = ON
+$EA	Word	Yes	True	T/F	Autoreceive Zmodem, True = ON
+$EC	String	Yes	'>> '		ReplyQuote string, 1 length word plus 16 characters
+$FE	Word	Yes	$49	$000A-$FFFF	Sendlines length value
+$100	Word	Yes	False	T/F	Use MacBinary on Forked files, True = ON
+$102	Word	Yes	False	T/F	Strip MacBinary on receive, True = ON
+$104	Word	Yes	$04	$00-$FF	Auto-receive default filetype
+$106	Word	Yes	$0000	$0000-$FFFF	Auto-receive default auxtype
+$108	Word	Yes	False	T/F	Use default Upload path on Send files
+$10A	Word	Yes	False	T/F	Use default Download path on Receive files
+$10C	Word	Yes	False	T/F	Use relaxed transfers, True = ON
+$10E	Word	Yes	False	T/F	Add times to debug statement, True = ON
+$110	Word	No	False	T/F	Script lock flag
+$112	Word	No	False	T/F	Serial-TCP/IP flag, True = TCP/IP
+$116	Word	Yes	True	T/F	Show service switch messages on screen and buffer
+$118	Word	Yes	$00	$00-$FF	Flush character used by TCP/IP
+$11A	Long	Yes	$00,$00	$00-$FF	EOL characters used by TCP/IP
+$11C	Word	Yes	$0200	$0000-$0200	Flush frequency used by TCP/IP
+$11E	GS-String	Yes	Empty		Signature string fired by OA-T menu item

ExtraVariables

Pointer to the block of settings that are stored when the script command Store Settings is used.

You may only directly alter those variables that are marked with a "Yes" as being write enabled. Be careful in changing even those values, as you may have to take some other action as well as just changing the value of the flag. **If in doubt, do not alter any of these values!**

The **Default** and **Range** entries show the default state of the flag. Where an entry has been made for **Range**, the range is also shown. A **T/F** entry for **Range** shows that there are only two states for the flag, **True** (non-zero) or **False** (zero).

Offset	Type	Write	Default	Range	Description
+$0C	Word	Yes	False	T/F	Flush to screen during scripts, True = ON
+$0E	Word	Yes	True	T/F	Screen on/off in scripts, True = ON
+$10	Word	Yes	False	T/F	Ymodem-g, True = ON
+$16	Word	Yes	True	T/F	Send Init flag, True = ON
+$1A	Word	Yes	$5E	$21-$7F	Caret equivalent used in scripts
+$1C	Word	Yes	$22	$21-$7F	Quote equivalent used in scripts
+$3C	Word	Yes	$01	$01-$03	TCP/IP data mode

ClockVariables

A pointer to a data block that can be used by status bars or clock displays. These values are generated by Spectrum and will change constantly.

```
ClockVariables   anop
TimerPaused      ds     2              true = timer paused, false = timer free running
ClockSeconds     ds     1              Current real time
ClockMinutes     ds     1
ClockHours       ds     1
                 ds     5              reserved
TimerSeconds     ds     1              Time when Timer was started
TimerMinutes     ds     1
TimerHours       ds     1
```

Displays should use the values in the **ClockVariables** data block for any clock displays they may show. Displays should not call any of the **Miscellaneous Tool Set** routines to read the system clock because they mask interrupts and therefore can cause character loss.

Note that the clock and timer values are single byte values and not the usual two byte words. The clock values will be either in 12 or 24 hour format depending on the format the user has selected. The timer values are always in 24 hour format.

Note that you should not change any of these values under any circumstances.

ScreenBuffers

A pointer to the **ScreenBuffers**, which are *two* arrays of 24 Pascal strings of 80 characters each (24*81 bytes=1944*2 = 3888 bytes). Each string represents a single row from the screen. The length byte of each string has a fixed value of 80. This must not be changed (fill the strings representing blank screen space with the Space ($20) character). The strings must only consist of ASCII displayable characters of value $20 upwards. Do not store any control characters in these strings.

The main purpose of the **ScreenBuffers** is to provide a smooth transition from one Online Display to another. While your display is open you can use the ScreenBuffers for any purpose you like, but typically a display would use the first 24-line buffer to store the text currently shown on the screen, while the second 24-line buffer would be used to store special attributes (color, mousetext, etc.) for each of those characters so that screen updates would be able to preserve those attributes.

ScreenPosition

A pointer to the cursor position (two words representing the Horizontal (0-79) and Vertical (0-23) screen positions). This is used to transfer the current position of the cursor between different displays.

```
ScreenPosition   anop
                 ds     4               Point representing cursor position.
```

Paths

A pointer to a block of GS/OS filenames and pathname buffers used by Spectrum. **Do not alter these buffers or their contents under any circumstances!**

```
Paths            anop

                 dc     i'38'
                 dc     i'11'           Filename of current Menu file
                 dc     c'Script.Menu'
                 ds     23

                 dc     i'38'
                 dc     i'6'            Filename of current script
                 dc     c'Script'
                 ds     28

                 dc     i'512'
                 dc     i'18'           Path of current script
                 dc     c'@:Spectrum.Script:'
                 ds     490

                 dc     i'38'
                 dc     i'8'            Default download filename
                 dc     c'Download'
                 ds     26

                 dc     i'512'
                 dc     i'10'           Default download path
                 dc     c'@:Download'
                 ds     498

                 str    'Spectrum'      Default font name
                 ds     17

                 dc     i'512'
                 dc     i'11'           Default autosave pathname
                 dc     c'@:SP.Buffer.1'
                 ds     495

                 dc     i'38'
                 dc     i'8'            Default upload filename
                 dc     c'Upload'
                 ds     28

                 dc     i'512'
                 dc     i'10'           Default upload pathname
                 dc     c'@:Upload'
                 ds     500
```

Palettes

A pointer to the 8 color palette tables for use within 320 mode displays.

Most emulations using the Super HiRes display will use the standard color palettes for the screen resolution that they have chosen. However some emulations may choose to build custom palettes to suit a custom display. Spectrum provides for an integral custom palette which uses 8 basic colors and a second 8 colors that flash on and off once per second. This palette can be seen in use in the Viewdata display.

To use this color palette feature you must first turn it on by setting Bit 14 of the A register when you return from the **Init_Driver** call, and also install the **Fixed palette** as the current color table. Spectrum will then automatically switch the Fixed and Flashing palettes once per second while your display is open.

The eight colour palette:

```
Fixed palette    dc     i'$000,$f00,$0f0,$ff0,$00f,$f0f,$0ff,$fff'
                 dc     i'$000,$f00,$0f0,$ff0,$00f,$f0f,$0ff,$fff'
Flash palette    dc     i'$000,$0ff,$f0f,$00f,$ff0,$0f0,$f00,$000'
                 dc     i'$000,$f00,$0f0,$ff0,$00f,$f0f,$0ff,$fff'
```

This Flash Palette lets colors flash to their opposite color. So that the system cursor itself does not flash, Black remains as Black and does not flash to White (but White does flash to Black).

The default palettes represents the following colors:

```
Fixed values 0-7       Black, Red, Green, Yellow, Blue, Magenta, Cyan, White
Flash values 8-15      Black, Red, Green, Yellow, Blue, Magenta, Cyan, White
```

It is permissible to alter the values in the two palettes if you wish, but only while your custom display is active. If you do change these values, you must restore the standard palette values as shown in the table above, when **Close_Driver** is called. If you do not restore the standard values, other displays may not show correctly on screen. Note that you must always switch on the palette feature by setting Bit 14 of the A register. You cannot stop Spectrum switching the two palettes once each second while this feature is activated.

ChatLineData

A pointer to a data block for use with the **ChatLine** feature (see Chat_Init and Chat_Display).

```
ChatLineData     anop
TextlineLength   ds     2
TexLineBuffer    ds     240
LinePosition     ds     2
```

FatalError

A pointer to the generic GS/OS error handling routine in Spectrum. Enter the call with the target GS/OS error number in the A register.

Example:

```
                 lda    #ErrorMessage    number of GS/OS error message
                 jsl    >FatalError
```

The actual screen display will depend on which kind of screen is currently being displayed (SHR screen shows standard alert window with OK button; Text screen shows text window with Press Escape message; Other screens show a text message only).

The following errors are displayed:

```
Err $07    str   'GS/OS is busy.'
Err $10    str   'Device not found.'
Err $26    str   'Resources not available.'
Err $27    str   'I/O Error.'
Err $28    str   'No device connected.'
Err $29    str   'Driver is busy.'
Err $2b    str   'Disk write protected.'
Err $2e    str   'Disk switched.'
Err $2f    str   'Device offline or no media.'
Err $40    str   'Bad pathname syntax.'
Err $43    str   'Invalid reference number.'
Err $44    str   'Subdirectory not found.'
Err $45    str   'Volume not found.'
Err $46    str   'File not found.'
Err $47    str   'Duplicate file name.'
Err $48    str   'Volume full.'
Err $49    str   'Volume directory full.'
Err $4a    str   'Version Error.'
Err $4b    str   'Unsupported storage type.'
Err $4c    str   'End-of-file encountered.'
Err $4d    str   'Position out of range.'
Err $4e    str   'Access not allowed.'
Err $50    str   'File is busy or already open.'
Err $51    str   'Directory Error.'
Err $52    str   'Unsupported volume type.'
Err $53    str   'Invalid parameter.'
Err $54    str   'Out of memory.'
Err $57    str   'Duplicate volume online.'
Err $5b    str   'Illegal pathname change.'
Err $5c    str   'Not an executable file.'
Err $5d    str   'Operating system not supported.'
Err $61    str   'End of directory reached.'
Err $81    str   'Out of memory for this operation.'
Err $82    str   'Tool Error.'
Err $83    str   'Cannot copy directories.'
Err $84    str   'File is empty.'
Err $85    str   'Cannot resume this transfer.'
Err $86    str   'Path not found.'
Err $87    str   'One or more files are not available.'
Err $88    str   'AppleTalk is using that slot.'
Err $89    str   'No serial card found.'
Err $8a    str   'Slot must be set in Control Panel.'
Err $8b    str   'Could not select a serial port.'
Err $8c    str   'File already exists.'
```

Those error numbers that do not have a linked message will display a generic message showing the error number in Hex format.

ScreenValid

A flag word which shows if the **ScreenBuffer** is filled with valid data (if **ScreenValid** is True then **ScreenBuffer** is valid). The **ScreenBuffer** is filled when a display shuts down with the **Close_Driver** call.

Note that this is an actual value and not a pointer. Only alter this value after you have filled the **ScreenBuffer** with valid data.

Switched

A flag word that tracks if **The Manager** has switched the Spectrum application in or out.

```
Switched = true  The Manager has reported TMSaysAppSwitchedOut for Spectrum
Switched = false The Manager has reported TMSaysAppSwitchedIn (the default state) for
                 Spectrum
```

Note that this is an actual value and is not a pointer. Do not alter this value under any circumstances.

SwitchKey

A flag word that tracks if **The Manager** has seen the **OpenApple-Control-Tab** keypress.

```
SwitchKey = true The Manager has reported TMSaysSwitchKeyPressed
SwitchKey = false      Default state
```

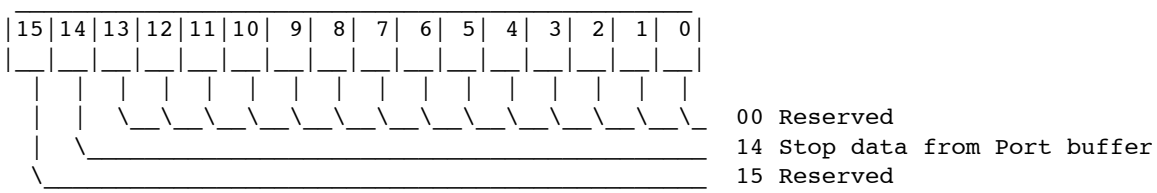
Note that this is an actual value and is not a pointer. Do not alter this value under any circumstances.

Freeze

A flag word which Spectrum uses to stop data being sent to the screen when direct screen writes might overwrite open windows.

You may optionally set Bit 14 to stop data being retrieved from the Port Buffer. This will only control the standard loop that Spectrum uses to send data to the display. You must then clear this bit to resume data input once more. Bit 14 will be automatically cleared when the `Close_Driver` call is made or if ⌘-Control-X is used to reset the port.

You ***must not*** change any of the other bits in this flag word and must preserve them through any access you may make.



LastRefCon

Before Spectrum closes a display it copies the **RefCon** of that display into this flag word. Displays can compare this value against their own **RefCon** to determine of they were the last display that was open. If the value does not match, they will then know that they are being called for the first time.

Note that this is an actual value and is not a pointer. Do not alter this value under any circumstances.

ScriptActive

If a script is currently active, this flag word will return a true value. Online displays can check this value and perhaps show an indicator on the screen.

```
ScriptActive = true     Script is currently active
ScriptActive = false    No script is currently active
```

Note that this is an actual value and is not a pointer. Do not alter this value under any circumstances.

DirtyEdit

When you `Close_Display` is called to close a custom Editor, online displays set this flag word if changes have been made to the Edit record that have not saved to disk. Clear this flag if no changes have been made since the file was last saved. Do not change this flag if you have not altered any of the values in the **EditBlock**.

Note that this is an actual value and is not a pointer.

ScreenMarker

When a display uses or alters the second block of **ScreenBuffer** data it sets this variable to its own **RefCon** so that it will know whether the data was written by itself or another display.

Note that this is an actual value and is not a pointer.

DirectSendVector

A pointer to the main serial port output call. No conversion or other processing is made by this call.

Example:

```
lda    Data             Data to output
jsl    >DirectSendVector
```

On return from the **DirectSendVector** call the carry flag will be clear if the data was successfully transmitted. If the data was not transmitted successfully, an internal routine within Spectrum will have attempted to handle the problem by repeatedly trying to transmit the data. If this still fails to transmit the data, the continuous "try to transmit" loop will appear to have caused a program hang. If the user then aborts this loop by pressing OA-Period or ESCape the **DirectSendVector** call will return with the carry flag set.

SetRTS OFF

A pointer to the address of a call which will turn RTS OFF in the Serial Port Driver if Hardware Handshake has been turned on. It has no effect if Hardware Handshake is not checked or turned on from a script. *IMPORTANT: You must not leave RTS turned off for too long, or you will lose data coming in to the port. Also, the modem used must be able to handle its flow control using a standard Hardware Handshake cable, and should have an internal buffer to hold incoming data while you have RTS set to OFF, otherwise data will still be lost.*

SetRTS ON

A pointer to the address of a call which will turn RTS back ON.

ClockDisplay

This is intended for use mainly by External Commands.

A pointer to the address of a function which displays the status bar clock if it has been turned on. This call will always display on the two SHR screens or the Spectrum Text screen, if the status is on. It will only display on custom displays if they support a clock display.Sample call:

Example:

```
                    jsl    >ClockDisplay
```

ScreenDisplay

This is intended for use mainly by External Commands.

A pointer to the address of a function which sends the character in the accumulator to the current screen display, via the online display's `Screen_Display` function. The character is processed by the display according to the displays characteristics. On return, the character may therefore have been processed or filtered by the display. If there is no display open, the call will return and do nothing.

After calling `ScreenDisplay`, call `BufferWrite` if the A register is not zero, then call `ScrollWrite` (always).

Example:

```
                    lda    Data             Data to display
                    jsl    {ScreenDisplay}
                    beq    SkipBuffer
                    jsl    {BufferWrite}
SkipBuffer          jsl    {ScrollWrite}
```

CursorDisplay

This is intended for use mainly by External Commands.

A pointer to a function which displays the cursor on the current screen. Always call this function when you are finished with a series of **ScreenDisplay** calls.

Example:

```
                    jsl    {CursorDisplay}
```

BufferWrite

This is intended for use mainly by External Commands.

A pointer to the address of a function which stores the data from the accumulator into the Capture buffer/file if capturing is on. The value in the accumulator is preserved round the call.

Example:

```
                    lda    Data             Data to capture
                    jsl    {BufferWrite}
```

ScrollWrite

This is intended for use mainly by External Commands.

A pointer to the address of a function which stores the data in the accumulator to the **ScrollBack** buffer. The value in the accumulator is preserved round the call. All port data should be sent to the **ScrollWrite** vector as the user might have the "Raw ScrollBack" option selected.

Example:

```
lda    Data             Data to capture
jsl    {ScrollWrite}
```

ScreenReference

Holds the RefCon of the currently displayed screen. If this value is zero, then no display is open.

InternalSysBeep

This is intended for use mainly by External Commands.

A pointer to the Spectrum Internal SysBeep call. This is interrupt friendly and respects the state of Carrier Detect etc.

Example:

```
jsl    {InternalSysBeep}
```

ScriptHandles

This is intended for use mainly by External Commands.

A pointer to a 40 byte block of ScriptEditorHandles (10 x 4). If an entry is zero, then there is no scipteditor active. They are numbered consecutively from 0-9.

An **XCMD** can reference these Handles directly from this block. It is also permissable to create, change or remove a Handle from this block as every script command that uses these Handles will pick up the correct Handle when the script command is first called. If a Handle is removed it should be properly disposed of and the entry zeroed out. The Handles are only valid while a script is running, and only create a Handle while a script is ruuning. The Handles are all killed when a script stops.

DumpPrint

Spectrum handles OA-Shift-3 and OA-Shift-4 internally for both the online displays and the **Spectrum Editor**. Some displays which use a **TextEdit** control, such as the **Browser XDisplay**, may not respond to these keypress sequences. This routine points to the internal **Spectrum** routine that handles this for you.

On entry:	X = EventMessage Y = EventModifiers
Where:	OA-Shift-3 dumps the screen to disk OA-Shift-4 prints the current window OA-CapsLock-Shift-4 prints the whole screen

This vector is only in Spectrum 2.2 or onwards.

Rows_Columns

Two words giving the maximum rows and columns that this display can handle. This could be useful for later additions to Spectrum. As this is a new parameter in v2.2 of Spectrum, please note that not all existing displays may respect this value.

This vector is only in Spectrum 2.2 or onwards.

TCPIPData

This is intended for use mainly by External Commands.

A pointer to this block of data for use with **TCP/IP**:

```
TCPIPData         anop
TCPIPInitialised  ds    2               Boolean, are we in TCP/IP mode
TCIPOpen          ds    2               Boolean, are we connected
TCIPOnline        ds    2               Boolean, have we an active socket
CurrentPort       ds    2               Current active socket
XFerFlag          ds    2               Set during TCP/IP file transfers
```

Routines can examine the TCPIPInitialised word to see if Spectrum is currently set to TCP/IP or Serial mode. The other values can be of use to determine whether TCP/IP is online or not.

TCPIPFlushVector

This is intended for use mainly by External Commands.

A pointer to the routine that flushes any pending data to the current active socket.

Writing Spectrum Online Displays

Written by: Ewen Wannop November 1993
Updated by: Dave Hecker September 1998

This technical note describes the way that Spectrum communicates with its Online Displays.

General Information

Spectrum emulation display drivers are loaded character drivers that are used to display the screen output of Spectrum. They can choose to display on any one of the possible IIgs display screens. Both Spectrum and the display driver have a parameter block through which they can communicate information with each other. Data to be displayed on screen is passed to the display driver using the processor registers and the stack, restricting coding to assembly language (or C/Pascal with some very fancy footwork).

The display drivers are load files having a filetype of $BC and an auxiliary type of $4015 (Spectrum Online Display).

When Spectrum is launched it searches the *Spectrum:Add.Ons:Online.Displays* folder for any load files of the correct file/auxtype. Any matching file is loaded into memory and added to a list of available displays (see Settings/Online Display).

It is the responsibility of Spectrum to primarily handle the interface with each display driver and of providing the data which the display will show on screen. Spectrum will normally handle the TaskMaster loop as part of its housekeeping and menu selection routines.

It was primarily intended for these display drivers to be screen emulations, but it is possible for the display driver to take complete control of all events. This added feature gives a display driver the power to do be able to function in ways that would not normally be expected of a screen display, and that have not been included within the main body of Spectrum.

All driver calls must all be supported within the driver to at least the minimum level required by that call. All calls are made in full native mode and must be terminated by an RTL.

Operating Environment

When a display driver is first called through the `Init_Driver` call, the screen will be showing the full SHR 640 desktop. Whenever the `Close_Display` is called, the display driver must leave the screen with this SHR 640 desktop restored.

The display driver is responsible for returning information from some of its calls, and for processing data as and when required by Spectrum. It must respect the various call protocols listed in this document. Apart from this basic interface requirement, a screen display can do almost anything that it wishes.

When the Init call has first been made to a display driver, the display driver is able to take over complete control and become a complete application in its own right. The display driver normally reports that it has opened, and then Spectrum would direct data destined for the screen to it.

NOTE: It is possible for a display to close itself down completely on the return from the Init call, in which case Spectrum will ignore the vector from then on as if the display had never opened. In this case, no screen output would be possible until a normal screen display had been selected.

Under normal circumstances Spectrum handles the processing of data from the port. After some internal checking of this data for various script, file transfer and other triggers, it then sends this data to the screen through the `Screen_Display` vector. A display would normally process this data through Spectrum's `InputFilter` vector and then display the data as appropriate to its screen display.

Because a particular display may actually handle screen data in a way that may not be generally compatible with other displays, a secondary `Direct_Display` vector is used to display generic screen messages such as those generated by scripts. This vector must conform to a basic range of screen commands, regardless of the handling of its main `Screen_Display` vector.

Data that has been captured from the keyboard, and is being sent to the port, is passed through the `Key_Handle` routine. This allows custom displays to modify certain key presses if they wish, before they are transmitted to the port.

Other calls are made by Spectrum through the display. This allows custom displays to handle such events as Editors, Status bars, Clock displays and Chat Lines. These are optional items for a display and may be omitted if desired.

Although loadable display drivers of the Spectrum designated type would not normally have a resource fork, it is permissable to have one. If a driver accesses the resource fork from its own code, it should make sure that it preserves the current resource file and resource file depth around any calls it may make, thus leaving things as it found them.

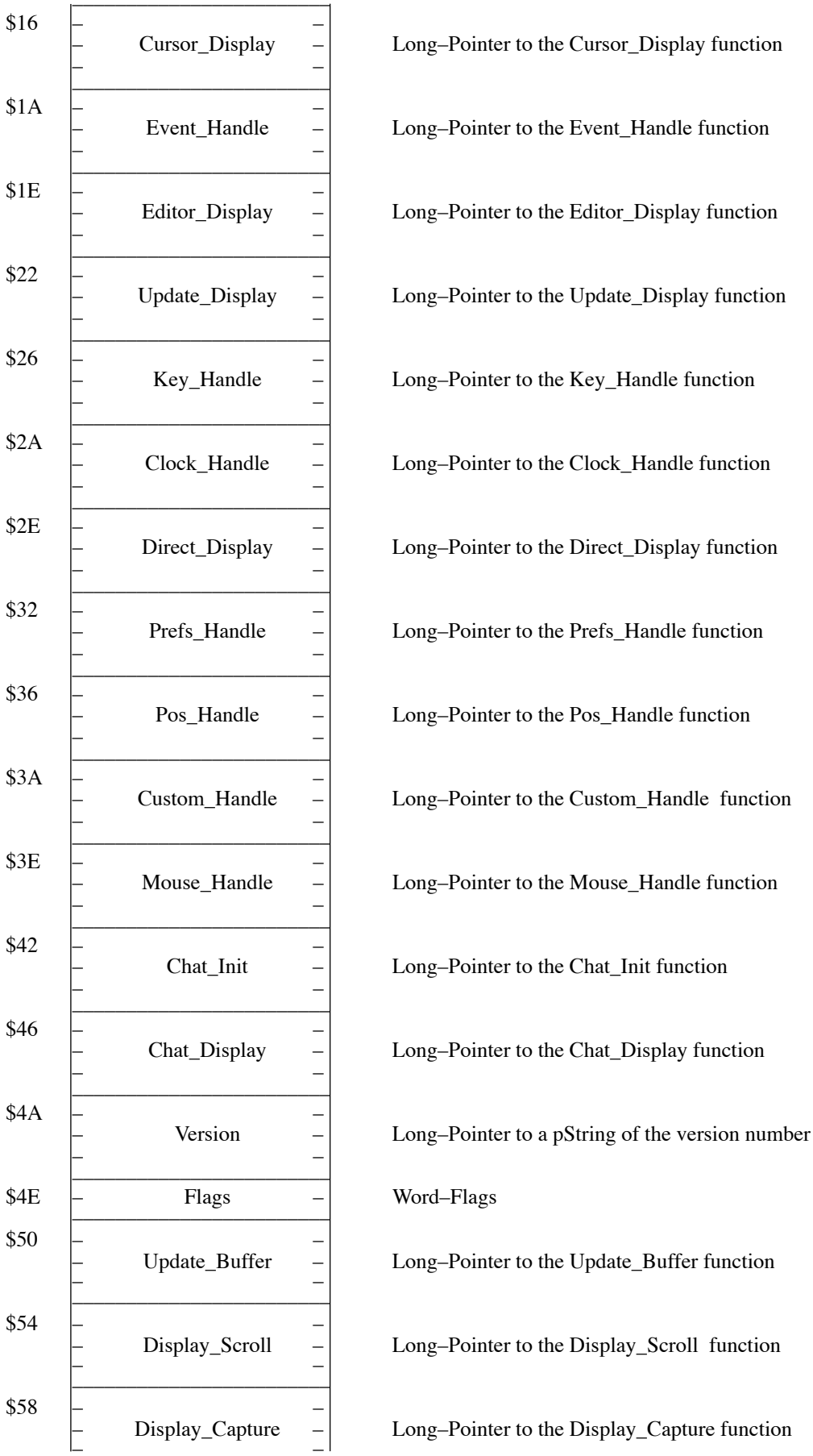
If you must turn off interrupts for any reason (highly **discouraged**, often there can be other ways) do so for the shortest time possible, and consider dropping/raising RTS round the interrupt mask.

If your Online Display creates any controls, it must use Control IDs in the range $7001 and above to avoid conflicts with Spectrum.

Driver Parameter Block

Spectrum requires each driver to have the following parameter block placed at its head:

$00	Marker	Long–Indentifying marker ("PICE")
$04	RefCon	Word–Unique reference number allocated by Seven Hills Software
$06	Title	Long–Pointer to a pString title
$0A	Init_Driver	Long–Pointer to the Init_Driver function
$0E	Close_Driver	Long–Pointer to the Close_Driver function
$12	Screen_Display	Long–Pointer to the Screen_Display function


$16
Cursor_Display
Long–Pointer to the Cursor_Display function
$1A
Event_Handle
Long–Pointer to the Event_Handle function
$1E
Editor_Display
Long–Pointer to the Editor_Display function
$22
Update_Display
Long–Pointer to the Update_Display function
$26
Key_Handle
Long–Pointer to the Key_Handle function
$2A
Clock_Handle
Long–Pointer to the Clock_Handle function
$2E
Direct_Display
Long–Pointer to the Direct_Display function
$32
Prefs_Handle
Long–Pointer to the Prefs_Handle function
$36
Pos_Handle
Long–Pointer to the Pos_Handle function
$3A
Custom_Handle
Long–Pointer to the Custom_Handle function
$3E
Mouse_Handle
Long–Pointer to the Mouse_Handle function
$42
Chat_Init
Long–Pointer to the Chat_Init function
$46
Chat_Display
Long–Pointer to the Chat_Display function
$4A
Version
Long–Pointer to a pString of the version number
$4E
Flags
Word–Flags
$50
Update_Buffer
Long–Pointer to the Update_Buffer function
$54
Display_Scroll
Long–Pointer to the Display_Scroll function
$58
Display_Capture
Long–Pointer to the Display_Capture function

$5C	Rows_Columns	Two words – Maximum rows and columns of this display

Marker

The identifying header marker for a Spectrum display consisting of four bytes in ascending order.

```
dc     c'PICE'
```

RefCon

A unique identifying **RefCon** to differentiate the various displays. A value for your display driver will be allocated by Seven Hills Software on request.

Title

A pointer to a pString representing the name of this Online Display (e.g. "Spectrum SHR Fast"). The maximum title length is 32 characters, but care should be taken to make sure the title fits nicely in the Settings/Online Display dialog box.

Init_Driver

The `Init_Driver` function may be called more than once. It is not always balanced by a `Close_Driver` call each time it has been called. For instance it will be called whenever the user chooses New from the File Menu. You should keep a local flag so you do not repeat the full initialization sequence if you are being called for a second time.

The first time this function is called the 640 SHR desktop (with menu bar) will be showing. You should perform any necessary screen preparation that you may require to display your screen.
NOTE: Shadowing is switched off. If you intend writing directly to screen memory you should first identify where the screen memory is before you start to write to it. Do not assume that it is always in bank $E1.

If you change screen resolution you must shutdown QuickDraw QDAuxilliary. You should get QuickDraw's direct page using the _GetWAP call and use this value when you start up QuickDraw again. If you change screen resolution from 640 mode to 320 mode you should preserve the mouse position across this change (e.g. divide the horizontal position by 2).

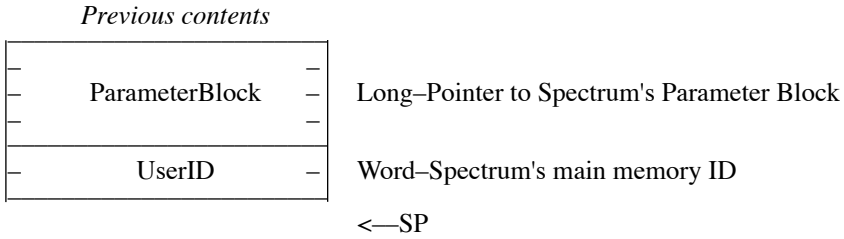
IMPORTANT: Even if you take over the entire screen and perform direct screen writes, you should still create a dummy window so that any mouse clicks will be seen as going into your window. This is most important for The Manager compatibility.

On the first `Init_Driver` call you should initialize any flags that may be needed by your other driver functions, such as the `Filter_Vector` or `Key_Vector` functions.

After your display has been opened you should update the screen using the information stored in the **ScreenBuffers**. If you use the second of the 24-line **ScreenBuffers** for attribute preservation, compare the **ScreenMarker** variable against your **RefCon** to determine if the attribute data was prepared by you. If so you can rely on the data and update the screen accordingly; otherwise you can only update the screen (as best you can) using the data in the first of the 24-line **ScreenBuffers**.

ON ENTRY the data is as follows

Stack



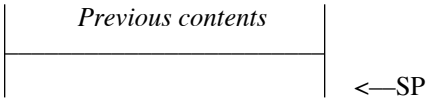
You must remember the userID passed to you and use this for any Handles that you need to create.

A register

A Word that identifies the key shortcut you should recognize for closing your online display (normally this will be ⌘W).

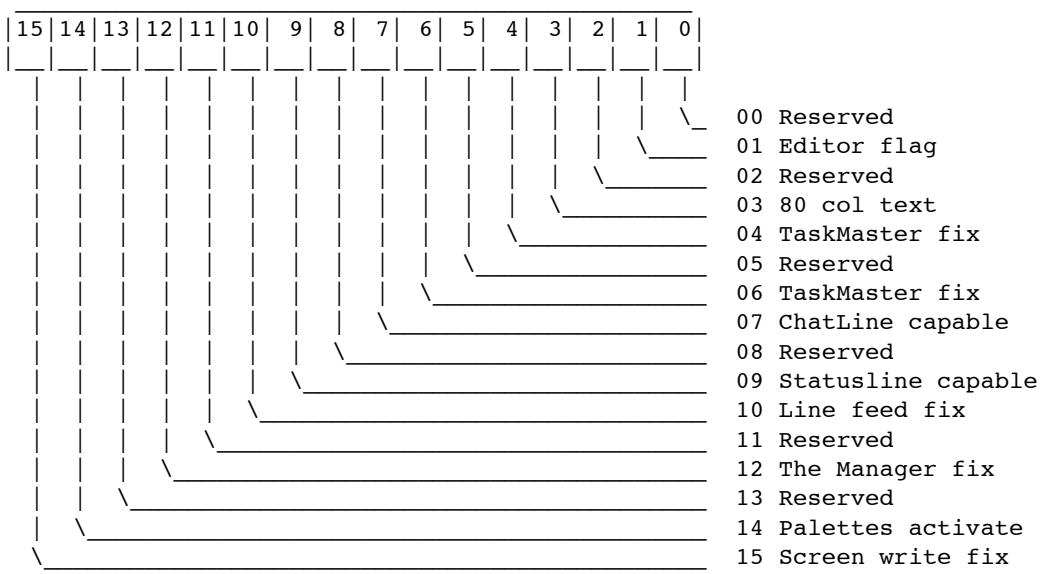
ON EXIT the data must be as follows

Stack



A register

Tells Spectrum how to behave towards your display.



Bits 0, 2, 5, 8, 11, and 13 are reserved and must be clear.

Bit 1: Set when an online display's editor has been opened. Clear for all others.

Bit 3: Set if standard 80 column text window is in use. Clear for all others.

Bit 4: Set to tell Spectrum to put $FFFF into EventMask and $00000002 into TaskMask. *NOTE: Do not set Bit 6 if Bit 4 is set*.

Bit 6: Set to tell Spectrum to put $FCF9 into EventMask and $00000000 into TaskMask. *NOTE: Do not set Bit 4 if Bit 6 is set.*

Bit 7: Set if your display supports a chatline. Clear if it doesn't.

Bit 9: Set if your display supports a status line. Clear if it doesn't.

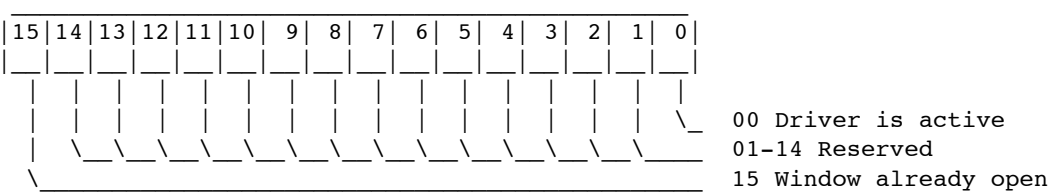
Bit 10: Set if you do not want linefeeds stripped from the input stream, even if the RemoveLFs option is on.

Bit 12: Set to prevent *The Manager* from switching out of Spectrum while your display is open. Clear if your display can operate in the background.

Bit 14: Set if you want Spectrum to automatically switch palettes approximately once per second. Clear for no automatic palette switching.

Bit 15: Set if you can operate with other windows in front of yours. Clear if your display writes direct to screen memory without first checking _FrontWindow.

X register



Bit 0: Set if you just now opened your display. Clear otherwise.

Bits 1-14: Reserved and must be clear.

Bit 15: Set if your display was already open. Clear otherwise.

NOTE: If Bits 0 and 15 are both clear then the display is ignored and all output vectors are made inactive.

Close_Driver

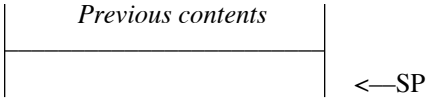
This function must close your custom screen. Some tasks you must perform:

- update the first of the 24-line **ScreenBuffers** (if necessary) so it contains an image of the text on your display
- update the second of the 24-line **ScreenBuffers** (if necessary) so it contains any information you desire (typically used for preserving screen attributes)
- store your current cursor position into **ScreenPosition**
- store your **RefCon** into the **ScreenMarker** variable
- set the **ScreenValid** flag to True
- If you are closing a custom Editor, and you have data in private editing buffers, transfer the Handles for your editor buffers into the **EditBlock**, and set the **DirtyEdit** flag true. Also set the **SelectionStart** and **SelectionEnd** values to your current selection (if you do not have a selection in your text record, zero these values). If you have no data to transfer from your Editor record, do not change the handles within the **EditBlock**, or the **DirtyEdit** flag.
- dispose of any Handles you have created

- restore the standard color palettes (if you changed them)
- return to show the standard 640 mode SHR desktop (with the system menu bar showing). If you had changed screen resolution you should preserve the mouse position across this change. Normally this would mean doubling the horizontal position if you are reverting to 640 resolution from a 320 screen.

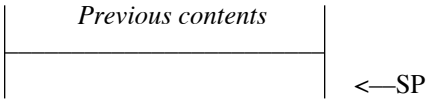
ON ENTRY the data is as follows

Stack



ON EXIT the data must be as follows

Stack



Screen_Display

This is the main vector through which Spectrum passes incoming data to be displayed on your screen. For this reason this code should be as "tight" as possible in order to make the screen display as quick as possible.

Data is checked for auto-receive, scripting triggers, and stored within the **Scrollback** buffer (if active and ScrollData is Raw), but it has not been processed in any other way. Most screen displays should immediately pass the character through the **InputFilter** routine, which processes the data according to the various pre-set filters that are currently active.

How (and whether) you actually interpret and display the character on the screen is entirely up to you and your display's protocol. If your display only recognizes 7-bit characters, after filtering you may wish to check to see if a high bit is still present and, if so, perform a simple-but-intelligent character translation (e.g. convert “ and ” to ", ‘’ to ', ® to R, © to C, é to e, etc.).

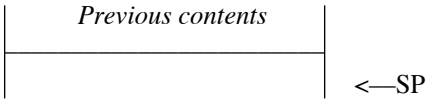
If you write directly to SHR screen memory you should refer to the various Apple Technical notes on this subject. It is important that you first check where screen memory lies and only write to that bank. Various utilities, such as Easy Access, may well move the screen memory from the normal $E1 bank to some other place in memory. For compatibility you can verify that _FrontWindow is your window and perform direct writes if so, or use safe QuickDraw calls if not; for speed you might skip the `_FrontWindow` check yourself (if you do this, see the `Init_Driver` function, Exit information, A register/Bit15).

*SPEED TIP: For speed you should display the given character at the current cursor position (wiping out any cursor currently shown) then calculate the new cursor position (scrolling the screen as necessary) but do **not** draw the cursor itself. Instead just clear a flag to indicate the cursor is not currently displayed (see Cursor_Display).*

Control codes that mean something to your display would also be interpreted here. For example, if your display contains codes that turn on or turn off the cursor, those codes should set/clear a flag which you can refer when you need to know whether it's OK to show the cursor.

ON ENTRY the data is as follows

Stack

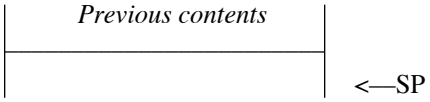


A register

Contains the (full 8-bit) character to be displayed. Usually you would immediately filter this character (`JSL >InputFilter` see the Spectrum Parameter Block for more information) before displaying it. If the Carry flag is clear then the character has been converted and should be displayed. If the Carry flag is set then the specified character should be ignored (though you can still examine the character to see if it is a linefeed, which you may require so you could still act on it).

ON EXIT the data must be as follows

Stack



A register

The character to store in the **Capture** buffer/file (if active), and in the **Scrollback** buffer (if active and ScrollData is Filtered). Return $00 if this character should not be stored. Screen position commands sent to VT100 screens for instance, should be stripped from the return stream, by changing returning $00.

Cursor_Display

This call is made whenever no port data is pending, and also after a Return ($0D) character has been processed. By only drawing the cursor at these times, port data will be displayed to the screen as quickly as possible.

If you don't support a cursor, or it is already displayed, then just RTL.

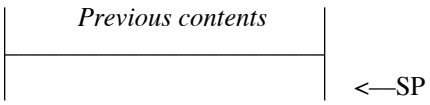
If you;

- do support a cursor, and
- it has not been switched off (by a "private" control code sent through Screen_Display, or by ^Fbeing sent through Direct_Display), and
- it is not already displayed on the screen,

then immediately draw the cursor, set a flag indicating it is now displayed, then RTL.

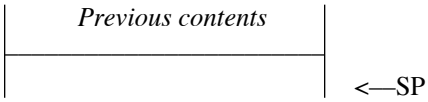
ON ENTRY the data is as follows

Stack



ON EXIT the data must be as follows

Stack



Event_Handle

Data in: A register, X register and Y register

Most system events reported by TaskMaster are handled directly within Spectrum itself. In addition, custom displays may well have turned off certain events (see the `Init_Driver` function, Exit information, A register/Bits 4 & 6). Remaining MouseDown, Key and Click events are passed through the `Event_Handle` vector for the driver to handle as it wishes.

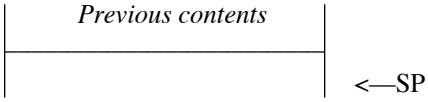
Many key events are handled by Spectrum, even if TaskMaster is ignoring Menu events, and will not be passed through to this call: D (dial number), E (show editor), G (show online display), Q (quit), M (send clipboard to modem), V (paste clipboard), W (close), H (hangup), B (send break), ` (toggle chatline), R (run/stop a script), * (re-run script in memory), I (send text file), J (send editor text), </, (start/stop timer), = (toggle capture), N (new), P (print), \ (show scrollback), | (show capture), Space (toggle status line), and ^X (reset port driver).

If the "Recognize Script Keys" option is checked, only Option-Delete will make it through to `Event_Handle`; other Option-KEY events will be eaten by Spectrum. If the option is not checked then all Option-KEY events will make it through.

Only *unclaimed* click events will be passed through. If you have created a control, and are expecting events, check _FrontWindow, then your Control ID, to verify the control is really yours.

ON ENTRY the data is as follows

Stack



for MouseDown events
A = EventWhat = $01
X = EventWhere+2
Y = EventWhere

for Key events
A = EventWhat = $03
X = EventMessage low byte
Y = EventModifiers

for Click events
A = EventWhat = $21
X = ControlValue
Y = NA

ON EXIT the data must be as follows

Stack

Editor_Display

When **Editor** is selected from the Spectrum menus, the `Editor_Display` function is called for you to open a custom editor. The call is only made if a custom Editor is supported by your display (see Driver Parameter Block, Flags, Bit 15).

`Editor_Display` should establish the screen environment for your custom editor (see `Init_Driver` for more information). When the custom editor is closed the `Close_Driver` function is called, just as it is with an online display. You must keep flags if you need to distinguish between the two screens you may have open.

On the first `Editor_Display` you should initialize any flags that may be needed by your other driver functions, such as the `Filter_Vector` or `Key_Vector` functions.

You should not use the **ScreenBuffer** for storage when in your custom Editor, this data buffer should be used only for online screens, never for an Editor.

Spectrum keeps a data block pointed to from within the **Spectrum Parameter Block** for use by Editors. See the **EditBlock** entry for further details. When Spectrum closes the standard SHR Editor, any data in the TextEdit record for the **SHR Editor** is transferred into the Handles shown in this block. On entry to your custom editor routine you can check to see if there are valid handles in the EditBlock and, if so, you can use this data as you wish.

If you modify the contents of these handles, make sure you update the various pointers correctly as you change the data! It is recommended that you copy the **EditBlock** Handles into your own data block, and zero the edit Handles in the **EditBlock** while you have control within your Editor. You can then wait until your editor display is closed to update the **EditBlock** entries. The only drawback to zeroing the handles in this manner is that the "Run Editor As Script" menu command will fail while in your custom editor.

If you require user interaction in your custom Editor, Mousedown events and KeyPresses are passed to your Editor through the `Event_Handle` function.

See the `Init_Driver` documention for complete information on the data that is passed to and from the `Editor_Display` function. Note that Bit 1 of the A register must be set on exit.

Update_Display

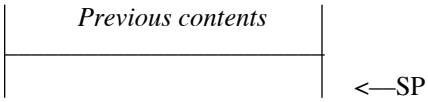
The `Update_Display` function is called frequently by TaskMaster and Spectrum. You should redraw your window whenever this call is made.

If you are displaying an 80 column or custom window, there will be no other windows open on top of yours other than any you have opened yourself. If you are displaying a standard SHR window on the desktop, be aware that there may well be other windows open on top of yours when this call is made, so this function should only draw using standard QuickDraw commands.

If you have opened a window using the **_NewWindow** or **_NewWindow2** calls, you should pass the address of the `Update_Display` call to **wContDefProc** in the **NewWindow** parameter list so that TaskMaster can call your update routine.

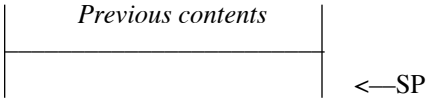
ON ENTRY the data is as follows

Stack



ON EXIT the data must be as follows

Stack



Key_Handle

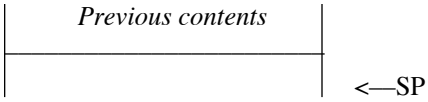
Spectrum passes all key presses (and simulated key presses) to the `Key_Handle` function. This function normally just sends the character in the A register on to the serial port output function:

```
Key_Vector    JSL    SendVector
              RTL
* OR for speed:
Key_Vector    JML    SendVector
```

However, it *is* possible for the Online Display to handle the key press in some other way. For example, certain key presses could be ignored completely, while other keypresses might be translated to some other key equivalent before being sent out to the port, and still other keypresses might cause several characters to be sent out to the port.

ON ENTRY the data is as follows

Stack



A register

The "processed" EventMessage, as modified by Spectrum's options (DeleteKey, send LineFeeds, OutputFilter, low ASCII, etc.).

X register

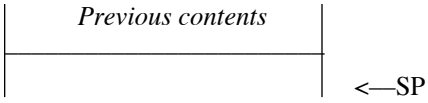
The EventModifiers (Note this will be $00 if a LineFeed is being sent after a CR with the ‘Set LineFeeds ON’ option).

Y register

The "raw" EventMessage, as returned by the Event Manager

ON EXIT the data must be as follows

Stack



A register

The character that was sent to the port, in case Spectrum needs to capture it (use #$00 if no character was sent to the port, and in cases where a single character caused several characters to be output to the port).

If you do not call SendVector, `CLC` before `RTL` to simulate a successful character send.

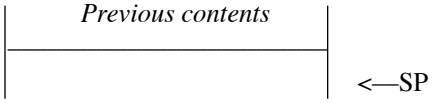
Clock_Handle

This function is not called if the SHR desktop with menu bar is visible. Under other circumstances this function is called approximately once per second so that you may update any status bar or clock display (observing the state of the **ClockShow** and **StatusLine** variables…monitor the state of these variables so you can clear the status/clock display if the option is turned off, or setup the status bar/clock display if the option is turned on).

You should **not** call the IIgs toolbox routine to get the current time because it masks interrupts, possibly losing some incoming data. Instead, pick up the current clock time and online timer values from the **ClockVariables** data block in the Spectrum Parameter Block.

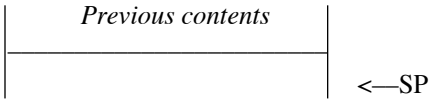
ON ENTRY the data is as follows

Stack



ON EXIT the data must be as follows

Stack



Direct_Display

When control codes are received by the main `Screen_Display` function, they will be interpreted differently by each online display (depending upon which particular terminal the display is emulating). The `Direct_Display` function is provided in order to give scripts a *consistent* way to display and manipulate data in the online display.

All displayable characters must be displayed to the screen using the U.S.A. character set, and the cursor must auto-wrap (when a character has been displayed in the rightmost screen column, set the column position to the far left column and increment the row position).

These control characters must be interpreted as follows:

Key	Result
^E	Show the cursor
^F	Hide the cursor
^G	Play the system beep sound
^H	Move the cursor one character to the left
^J	Move the cursor down one line
^K	Clear the screen from the cursor on down, without changing the cursor position
^L	Clear the display and move the cursor to the top-left of the display
^M	Move the cursor all the way to the left of the current line, then move the cursor down one line
^Y	Move the cursor to the top-left of the display without clearing the display
^]	Clear from the cursor to the end of the line
^^	GotoXY (set a flag to interpret the ASCII value of the NEXT TWO characters as X+32 and Y+32)

*NOTE: When you receive ^M set a flag that says "if the NEXT character is ^J, ignore it". Likewise when you receive ^J set a flag that says "if the NEXT character is ^M, just put the cursor all the way to the left; **don't** linefeed". This insures that a "^M^J" or "^J^M" sequence will not cause a double-space.*

Support for the following control characters are optional, but highly recommended:

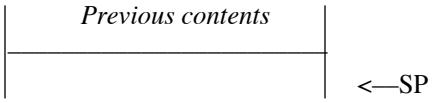
Key	Result
^I	Move the cursor right, to the next “tab stop” (usually at positions 0, 8, 16, 24, and so on)
^N	Normal text (turn off Inverse)
^O	Inverse text
^V	Scroll the screen down one line without changing the cursor position
^W	Scroll the screen up one line without changing the cursor position
^X	Turn off MouseText
^[	Turn on MouseText (display uppercase letters as “MouseText” characters)
^\	Move the cursor right one character
^_	Move the cursor up one line
^Z	Clear the entire line that the cursor is on

*NOTE: If it makes sense, your display can require Inverse **and** MouseText to be turned on in order to display MouseText (users are told to use the sequence ^O^[to insure MouseText is on, and to use the sequence ^X^N to insure MouseText is off).*

All other control characters should be ignored.

ON ENTRY the data is as follows

Stack

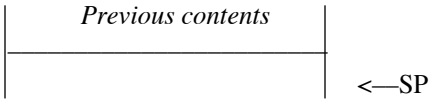


A register

The character to display.

ON EXIT the data must be as follows

Stack



A register

The character that you displayed.

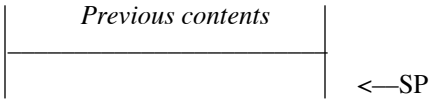
Prefs_Handle

Displays can show a preferences dialog box on the SHR screen. If you do not have a preferences dialog just RTL without changing the A register.

The SHR 640 screen with menu bar will always be active when this call is made with theA register being True. You should preseve the current port and present your preferences dialog in whatever format you desire. When your dialog is accepted, close it, restore the port, then RTL.

ON ENTRY the data is as follows

Stack



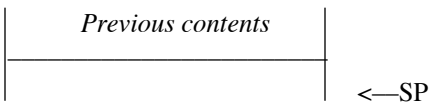
A register

If the A register is False on entry, return (in the A register) False if you do not support a preferences dialog, or True if you do support a preferences dialog.

If the A register is True on entry, display your preferences dialog box.

ON EXIT the data must be as follows

Stack



A register

If the A register was False on entry, return True or False to indicate whether you support a preferences dialog box.

If the A register was True on entry it doesn't matter what it is on exit.

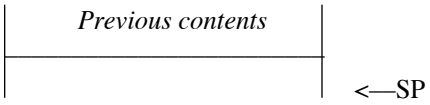
NOTE: From Spectrum 2.2 onwards, this call may be made before the Init_Driver call has been made. If you keep a 'preferences' file for your display on disk, you should open it from whichever call is made first so the settings are current and remain in force.

Pos_Handle

Spectrum calls this function when it needs to know where your current cursor position is located.

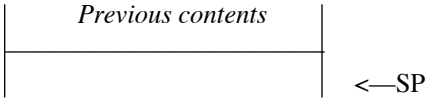
ON ENTRY the data is as follows

Stack



ON EXIT the data must be as follows

Stack



X register

Horizontal column (0-79)

Y register

Vertical row (0-23)

Custom_Handle

This special routine allows users to interact with a custom display in a way that is controlled entirely by the designer of the display. It is called by the following Spectrum Script command:

```
DirectAction "String to pass to the Online Display" variableName
```

When this call is made the A and X registers hold a pointer to a GS/OS string. The string consists of a length word followed by a string of up to 128 bytes (in the example above the string would have the length word of 36, followed by the 36 characters "String to pass to the Online Display").

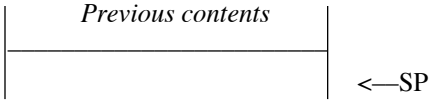
What you do with the string is entirely up to you and your display. Commonly it is used to pass private commands to a display (for examples see the on-disk documentation for the Spectrum SHR Fast display).

If you do *not* support any custom calls from a script just `RTL`, leaving the A and X registers intact. If you *do* support a custom call you should alter the string being pointed to (it's in a 128 byte buffer), point A/X to a new string, or store zero in A/X to represent a null string. *NOTE: If you create a memory Handle for the returned string you must dispose of this Handle yourself when you close your display.*

On return, Spectrum compares the returned string against the original string in the script. If the returned string is different, Spectrum stores the string into the variable name specified by the script author.

ON ENTRY the data is as follows

Stack



A register

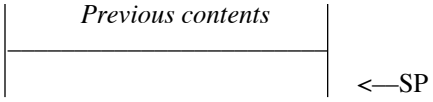
LoByte GS/OS String Pointer

X register

HiByte GS/OS String Pointer

ON EXIT the data must be as follows

Stack



A register

LoByte GS/OS String Pointer

X register

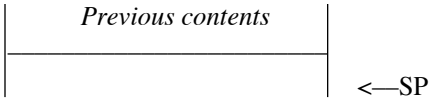
HiByte GS/OS String Pointer

Mouse_Handle

Spectrum calls the `Mouse_Handle` function when a mouse click position needs to be converted into a column/row position in your display.

ON ENTRY the data is as follows

Stack



X register

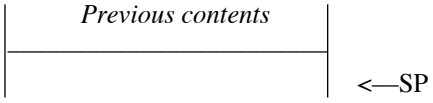
The mouse X position in local coordinates

Y register

The mouse Y position in local coordinates

ON EXIT the data must be as follows

Stack



X register

The horizontal column (0-79)

Y register

The vertical row (0-23)

Chat_Init

Spectrum supports an optional **ChatLine** display. An example of how a **ChatLine** display works can be seen in the SHR and 80 column text displays in Spectrum. A **ChatLine** display lets the user compose a response at the keyboard in a small edit area of the screen, and then send this data in a single continuous stream when the Return key has been pressed. The standard **ChatLine** consists of 3 lines of 80 characters, but it can be less than this.

Spectrum maintains a buffer for the **ChatLine** data. You must store the **ChatLine** data you receive into this buffer. Spectrum will use the data from this buffer when it sends the **ChatLine** to the port. The buffer is 240 bytes long and has a word length at the start. See **ChatLineData** in the Spectrum Parameter Block documentation for more information.

It is up to you exactly how you display the ChatLine on screen, but you must follow certain conditions both at **Chat_Init** and **Chat_Display**:

When you are passed the instruction to switch on your **ChatLine** display, you should clear any screen area you will need to use ready for the **ChatLine** to be displayed. This typically involves scrolling the screen up if the cursor position is currently in the area to be used for the chatline (for example, if the chatline will occupy rows 17-20 and the cursor is on row 18, only 2 lines would be scrolled so the cursor position would now be on row 16…if the cursor was on row 16 to start with no scrolling would occur).

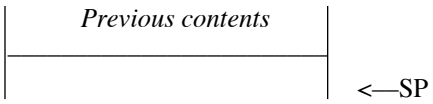
After you have prepared the screen, look at the **ChatLineData** data block and see if any data is pending in the **TextLineBuffer** (i.e. is the **TextLineLength** non-zero?). If there is data in the **TextLineBuffer**, you should display this data to screen in your **ChatLine** display.

The **LinePosition** pointer indicates where in the line the last display left its active writing position. You should set your active position to this point despite the actual length of **TextLineLength**. The **LinePosition** will never exceed the **TextLineLength**. It is up to your display to update **LinePosition** with its correct value when you close your chatline or your display. If your chatline is not editable **TextLineLength** and **LinePosition** should always be the same values.

When you receive the flag to switch off the **ChatLine** display, remove the display and restore the screen to its normal state. Do not change the **TextLineLength** or **LinePosition** word, and do not alter the data in the **TextLineBuffer** (so if the chatline gets switched back on the data will still be present).

ON ENTRY the data is as follows

Stack



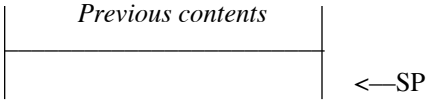
A register

If the A register is True on entry, initialize your chatline (if you have one), display any pending chatline data, and position the editing cursor.

If the A register is False on entry, turn off your chatline.

ON EXIT the data must be as follows

Stack



Chat_Display

If Spectrum has called `Chat_Init`, certain characters will then be directed to the `Chat_Display` function. You act on these characters, altering the **TextLineBuffer** and **TextLineLength** as appropriate. Any changes to the TextLineBuffer also must update the screen display. *NOTE: If the chat buffer is full you should play sound event #$A (Input Field Full) then set the Carry flag and RTL.*

If your chatline is active and you handled the character you must set the Carry flag before returning. If your chat display is not active (or your online display is currently closed) then the Carry flag should be cleared, which tells Spectrum to handle the data itself.

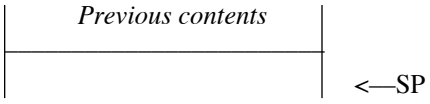
Your display will see only regular text characters ($20-$FF) and the following control characters:

Char	Action
$08	Backspace - Move the cursor one position to the left, stop at top left, adjust LinePosition
$0A	Line feed - move cursor down one line, stop at bottom, adjust LinePosition
$0B	Reverse line feed - move cursor up one line, stop at top, adjust LinePosition
$0D	Wipe your chatline display but **do not** zero TextLineLength…leave $0D in the A register and Spectrum will send the text to the port and zero the TextLineLength and LinePosition.
$15	Forward - Move the cursor one position to the right, stop at bottom right, adjust LinePosition
$18	Wipe your chatline display and store zero in TextLineLength and LinePosition
$7F	Delete the character to the left of the cursor, and pull any text beyond it back one position (on screen and in the TextLineBuffer…adjusting TextLineLength and LinePosition as appropriate)

If you are using dynamic editing within your **ChatLine**, you can use the **LinePosition** variable to point to the active position within your **ChatLine**.

ON ENTRY the data is as follows

Stack

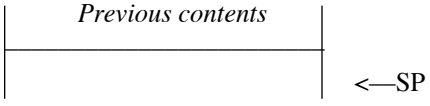


A register

Holds the character to process (see above).

ON EXIT the data must be as follows

Stack



A register

Should be unchanged from entry

Carry flag

Set if your display is active (the character was handled)

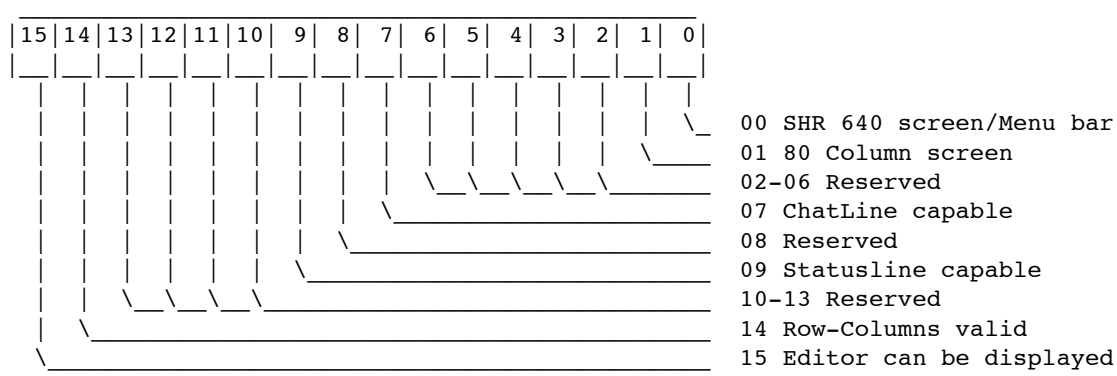
Clear if your display or chat display is not open (Spectrum needs to handle the character)

Version

The Version field points to a pString of the ASCII equivalent of a Long Version Format number (e.g. "v1.0").

Flags

A fixed flag which identifies the screen format of standard displays. Custom displays should only set bits 0-1 if they are using the standard SHR screen or the standard 80 column display.



Bit 0: Set if you are using a standard 640 screen with Menu bar.

Bit 1: Set if you are using the standard 80 column text display.

Bit 7: Set if ChatLine capable. (Spectrum 2.2 and onwards).

Bit 9: Set if StatusLine capable. (Spectrum 2.2 and onwards).

Bit 14: Set if Row-Column values are valid. (Spectrum 2.2 and onwards).

Bit 15: Set this if your display supports a custom editor. Clear otherwise.

All other bits are reserved and should be clear.

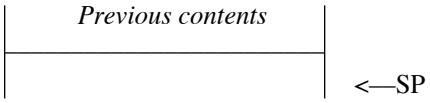
Update_Buffers

Spectrum may call this vector at any time. You should immediately transfer the contents of any private screen buffer to the global **ScreenBuffer**, in the same way that you would transfer the contents in the `Close_Driver` function, then clear the Carry flag and return.

If you do not maintain a private screen buffer simply return directly without changing the status of the carry flag. Spectrum enters this call with the carry flag set. If the flag is set, or has not been cleared on exit from this call, Spectrum will act as though no **ScreenBuffer** exists. For instance Spectrum makes this call when it dumps the screen to disk with the Shift-3 call. If the carry flag is cleared, showing the **ScreenBuffer** is valid, a text copy of the **ScreenBuffer** is dumped to disk as well.

ON ENTRY the data is as follows

Stack

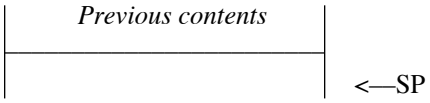


Carry flag

Is always Set on entry.

ON EXIT the data must be as follows

Stack



Carry flag

Clear if you updated the global **ScreenBuffer** to match your private screen buffer.

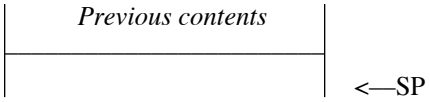
Display_Scroll

This function handles a custom Scrollback display (see **ScrollBuffer** in the Spectrum Parameter Block documentation). If you do not support this function, RTL immediately without changing the Carry flag.

If you do support this function, be aware it can be called whether or not your display is open so you must take appropriate steps to preserve and restore the current screen state as necessary (even closing the online display if necessary). If you handle the function you must clear the Carry flag before exiting, otherwise Spectrum will open its own Scrollback window.

ON ENTRY the data is as follows

Stack

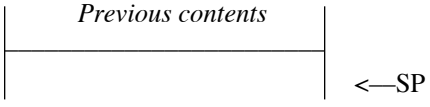


Carry flag

Is always Set on entry.

ON EXIT the data must be as follows

Stack



Carry flag

Clear if you successfully displayed the scrollback data.

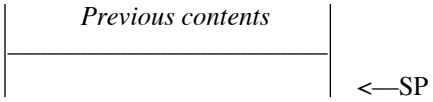
Display_Capture

This function handles a custom Capture display (see **CaptureBuffer** in the Spectrum Parameter Block documentation). If you do not support this function, RTL immediately without changing the Carry flag.

If you do support this function, be aware it can be called whether or not your display is open so you must take appropriate steps to preserve and restore the current screen state as necessary (even closing the online display if necessary). If you handle the function you must clear the Carry flag before exiting, otherwise Spectrum will open its own Capture window.

ON ENTRY the data is as follows

Stack

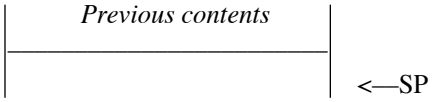


Carry flag

Is always Set on entry.

ON EXIT the data must be as follows

Stack



Carry flag

Clear if you successfully displayed the capture data.

Rows_Columns

Two words giving the maximum rows and columns that this display can handle. This could be useful for later additions to Spectrum. As this is a new parameter in v2.2 of Spectrum, please note that not all existing displays may respect this value. Check bit 14 of the Flags word to see if they are valid or not.

Within the folder '*:System:Desk.Accs:' create a folder called 'Help.Files:'. Within this folder create folders for each program for which there will be Help files available. The names of these folders are used to build the Subject popup menu, but if a text (or Teach) file called 'Subject.Alias' is found in the target folder, then the first 34 characters within that file are used for the Subject name instead. These folders would normally be created by an installer for the target program. Within each of these folders must be an 'Index' file constructed of separate lines in this way:

Start of file:
Nothing or blank lines at start
Each Topic entry:
MenuFlags (see list)
Name for the Topic Menu Item (34 max)
Full path to the file on disk (508 max)
At least one blank line to space to next item
Further Entries:
As above
End of file:
Optional blank line or lines to finish

Note that the path is normally a full path to the Help files. This means that the files can be anywhere on disk including the Help folder. To point to a file within the folder holding the Index file, it is necessary to use ':System:Desk.Accs:Help.Files:ProgName:'. However, if only a partial path is given, then the prefix ':System:Desk.Accs:Help.Files:SUBJECT:' is added.

The MenuFlags (these follow the normal Menu conventions):

N = normal (removes other flags)
P = normal (removes other flags)
S = shadow
O = outline
D = disable item
U = underscore
V = divider bar
I = italics
B = bold
X = XOR highlight

You can use multiple flags, but they will be acted upon in sequence, so an 'N' after a 'V' removes the effect of the 'V'.

When !Help! is opened from the Apple Menu, or called from an IPC call, it will build the Subject Menu from the folders and optional aliases it finds. It then attempts to set it to the current running program. If that fails, it is set to the first item on the list. The Topic Menu is then built from the Index file within the selected Subject folder, and the first entry is called up and displayed. If no Index file is available 'None Available' is displayed.

If !Help! is called a second time from the same program as before, it attempts to set to the same Topic and the same position within that Topic.

IPC Calls

A program can control the !Help! NDA by the use of IPC calls.

Send IPC calls to:

```
$8000 =  (to "Seven Hills~!Help!" stopAfterOne)
DataIn:
  dc i2'4'          number of parameters in this block
  dc i2'Action'
  dc i4'Subject'
  dc i4'Topic'
  dc i4'TargetString'
DataOut:
  dc i2'recvCount'
```

Action = 1 (Are you there). This call is simply accepted, no other parameters needed.
= 2 (Open !Help! NDA) Primary call.
= 3 (Close !Help! NDA) No other parameters needed.
= 4 (Open !Help! NDA) Help Menu call.

For the 'Open Action', command 2 only:

Subject = Pointer to pString, max 15, for a program/folder name within the Subject Menu.
The string is not case sensitive.

Topic = Pointer to pString, 34 max, for a Topic name within the Topic Menu. The string is case sensitive.
(Note that if the pString parses out to a valid decimal number, and the number is of a valid item in the Topic menu [range 1 to numitems] that item will be selected.)

TargetString = Pointer to a target string, 128 max. That string will be searched for and the text will be scrolled to the first occurrence of that string. The string is case sensitive.
(Note that if the pString parses out to a valid decimal number, the display will be scrolled to that point in the text, or to the end, if that is less than the decimal number.)

If any of the passed pointers are zero, or are not valid, then the default settings are used. This would be the current program, the first item on the Topic Menu, and the start of any text.

For the 'Open Action' command 4 only:

Subject = Pointer to a pString representing the name of the current application.
The string is not case sensitive.

Topic = The ID of the selected Menu (2 bytes).

TargetString = The ID of the selected Menu Item (2 bytes).

Note: Command 4 allows programs, through the use the _MenuGlobal option, to send the IDs of a selected Menu Item to !Help!. !Help! will pick up the name of the current application to select the 'target' folder. 'Subject' need only point to a suitable identifying string that might be displayed in later versions of !Help!.

!Help! will then use the passed Menu IDs to cross reference a file called 'Data' in the 'target' folder.

This file should be constructed as follows. The entries do not use quotes, and must be an exact match, with
no leading or trailing spaces. The entries within the file are similar to Command 2:

Start:
Empty line(s) to start
Entry:

$0000	MenuID, Hex value with '$' identifier
$0000	MenuItemID, Hex value with '$' identifier
Subject	Program/folder name within the Subject Menu (15 chars max)
Topic	Topic name within the Topic Menu (34 chars max)
TargetString	Search string or decimal position within file (128 chars max or decimal number)

At least one empty line to space to next item
Further Entries:
As above
End:
Empty line or line(s) to finish

Problems

Hopefully you will have none, but if you do, and they cannot be answered by reading these notes, please contact me on:

spectrumdaddy@me.com

Other information

Check for the latest version of Marinetti:

http://www.apple2.org/marinetti/

If you do not already know about my other software, please drop by my home pages and read more. Amongst other titles there you will find SAM2 an email client, SAFE2 an FTP client, and SNAP a Usenet news reader..

You will also find on my web site regular updates to my own programs, PDF manual versions to many of them, as well as many other files that you might find useful:

http://www.wannop.info/speccie/

Someone once said to me, 'Spectrum™ does everything!