

Dedicated to the memory of Joe Kohn 1947-2010

HTML Tool Set is Freeware and Copyright © 1996-2022 Ewen Wannop

HTML Tool Set and its supporting documentation may not be printed,
copied, or distributed for profit.

Distributing and/or archiving is restricted while in an electronic form.

Any “free” distribution must be given permission by Ewen Wannop
in advance -- please contact via email by sending mail to:

spectrumdaddy@speccie.uk

There is no guarantee that the right to redistribute this material
will be granted. The contents of this document may not be
reprinted in part or in whole.



Contents



Introduction	4
Using the HTML Tool Set	
Requirements	5
Programming	6
Using the Calls	8
The Routines	9
The Calls	
HTMLBootInit	10
HTMLStartup	10
HTMLShutDown	11
HTMLVersion	11
HTMLReset	12
HTMLStatus	12
HTMLInit	13
HTMLParse	14
HTMLOpenDisplay	15
HTMLCloseDisplay	16
HTMLInsertTERecord	16
HTMLClearDisplay	17
HTMLEventTask	17
HTMLGetTitle	19
HTMLGoToName	20
HTMLFindString	21
HTMLGetForm	22
HTMLCheckForForm	23
HTMLGetError	24
HTMLParse2	25
HTMLEventTask2	26
HTMLGetForm2	27
HTMLOpenDisplay2	28
HTMLEventToDisplay	29
HTMLGetPage	30
HTMLPostForm	34
HTMLCacheControl	38
HTMLSetTextTERecord	42
HTMLFindString2	43
HTMLExtractText	44
HTMLWriteToLog	46
HTMLSetLogPath	48
Appendix	
Cookies	49
Example of Posting a Form	50
Rolling your own Web Browser	51
Using the HTML Color & Custom tags	52
HTMLTool Set Error Codes & Links	54



Introduction



Introduction to the HTML Tool Set

Back in 1996, after a late night session at KansasFest, Geoff Weiss and myself took it upon ourselves to write an HTML Browser for the IIgs.

Geoff wrote the Spectrum scripts to make it all work, and I wrote an HTML engine that was used to capture and parse the HTML data to build the display. Spectrum Internet Suite, or SIS as it is better known was born.

Subsequently I used the source code from the engine to add HTML display features to Spectrum, more recently to add a custom HTML display to SAM2, and then adapted that original code to produce a stand-alone HTML Tool Set.

I have now expanded the HTML Tool to include calls that GET web pages and POST forms. This allows web browsers to be built with only a few Tool calls.

The HTML Tool Set has been released into the Public Domain, so that anyone who wishes, can build their own web browser, or display HTML within an application.

Full programming details for using the Tool are given within this manual.

It is up to you to do the rest!

If you have any questions about the use of this toolset, please contact:

spectrumdaddy@speccie.uk

HTMLTool Set is Freeware and Copyright © 1996-2022 Ewen Wannop



In Use



Using the HTML Tool Set

Requirements

The HTML Tool Set should be started after the application has started any other tools that it requires.

The HTML Tool Set requires these tools to be active:

Control Manager, Dialog Manager, Event Manager, Font Manager, Integer Math Tool Set, Memory Manager, Miscellaneous Tool Set, QuickDraw II, QuickDraw II Auxiliary, Scrap Manager, TextEdit Tool Set, Window Manager

Note: If you are using the calls HTMLGetPage or HTMLPostPage, your application should first check that version 1.0.3 or later of the HTML Tool is present, and that Marinetti TCP/IP 3.0b11 is installed and active.

The HTML Tool Set requires several custom fonts to be installed. These are supplied with the original archive:

SIS-1, SIS-2, SIS-3, SIS-4, SIS-5

In addition, the HTML Tool Set can use the ByteWorks Talking Tools, which are available on the **OPUS]]** collection from:

<https://juiced.gs/store/category/software/>

To obtain the HTML Tool Set, and any of my other software:

<http://speccie.uk>

Programming

The HTML Tool Set may be used in two ways, either as an engine to build a TextEdit Record from supplied text containing HTML code, or to further display a TextEdit Record in a window where the user can interact with the display.

In all cases where a TextEdit Record is returned, it is the responsibility of the application to kill the TextEdit Record when it has finished with it.

To use the HTML Tool Set, you should first start up any other tools you may require, making sure those required by the HTML Tool Set have also been started. To prime the Tool Set, call **HTMLInit**, checking that the call was successful.

To parse text containing HTML, call **HTMLParse** or **HTMLParse2**, and then on return from the call, either display the TextEdit Record within your own window, or within the optional window opened from the **HTMLOpenDisplay** or **HTMLOpenDisplay2** calls using the **HTMLInsertTERecord** or **HTMLSetTextTERecord** calls. For **HTMLOpenDisplay** the optional window is opened full screen, drawn just below the menu bar, and contains one TextEdit Control. For **HTMLOpenDisplay2** the window will be drawn according to the supplied Window Template.

The parsing process builds a display based on **HTTP/1.1** specifications. The IIGs display is fairly limited, so you will find that tables may not display correctly, also any embedded scripting will be ignored. The HTML Tool Set builds a page with various links and other hidden data drawn in a zero width font. This allows data to be contained invisibly within the display. This data can then be later retrieved from the **HTMLEventTask**, **HTMLEventTask2** or **HTMLEventToDisplay** calls when the user double-clicks a link.

For simpler applications such as the HTML display in the SAM2 email client, the TextEdit Record can be displayed within your own window, but if you intend to build a web browser, you will need to either use the optional window from the Tool, or one from a window template with the specified TEControl ID, for the user to have full control over the hidden data in the TextEdit Record.

If a URL or Link data is returned from the **HTMLEventTask** call, it is up to the application to verify the data is valid, what kind of data it is, and then to take the appropriate action.

If Form data is returned from the **HTMLGetForm** or **HTMLGetForm2** calls, it will be up to the application to take the appropriate action in posting the data.

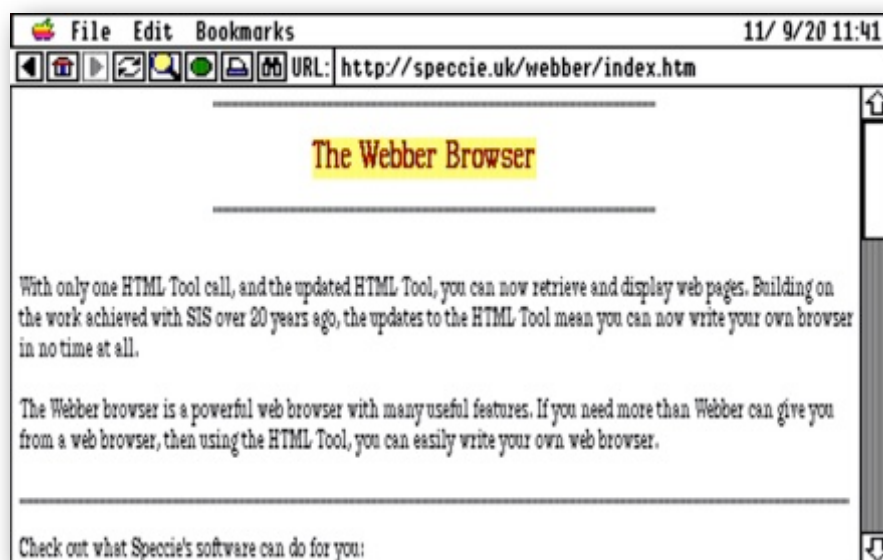
To build a complete web browser using the interactive window, you can retrieve web pages from a web server using the **HTMLGetPage** call. This requires TCP/IP to be installed and configured. From a URL passed in the call, this call can not only retrieve the raw data for the page, but optionally parse the data, returning a formatted TextEdit Handle, that can be inserted into the display window with **HTMLInsertTERecord** or **HTMLSetTextTERecord**.

If there are any Forms in the page, when their Save or Submit buttons are double-clicked, use the Form number returned from **HTMLEventTask** or **HTMLEventTask 2**, and pass it to **HTMLPostForm** to post the data.

When you request a page with the **HTMLGetPage** call, if it reports there was no `<html>` tag within the data, you may well have download a plain file instead. Along with the **HTMLGetPage** call set to return only HEAD data, the browser can use this information to download files from a web server, either automatically saving the data to disk from the Handle, or giving the User the choice of where to save the returned data using a Standard File dialog.

A web browser will need a considerable amount of code in addition to these few calls, but with just that one **HTMLGetPage** call, you can return a fully formatted TextEdit Record of a web page, thus taking much of the hard work out of building your own browser!

If you are creating your own web pages, you can always check how an HTML file you have built will display using the HTMLTool, if you open the file in the Spectrum™ Editor. If the file contains an `<html>` tag, you will first be asked if you wish to display it as HTML. Click "Yes" to see how it will display.



Using the Calls

Simple HTML Parsing

Call sequence:

```
HTMLInit
HTMLParse or HTMLParse2
HTMLGetError
```

Using the Optional Window

Call Sequence:

```
HTMLOpenDisplay or HTMLOpenDisplay2
HTMLInsertTERecord or HTMLSetTextTERecord
HTMLEventTask or HTMLEventTask2
HTMLEventToDisplay (optional)
HTMLCloseDisplay
```

While the window is open, these calls are active:

```
HTMLClearDisplay
HTMLGetTitle
HTMLGotoName
HTMLFindString or HTMLFindString2
HTMLGetForm or HTMLGetForm2
HTMLCheckForForm
HTMLExtractText
```

Receiving web pages or files

To retrieve pages:

```
HTMLGetPage
```

To post forms:

```
HMTLPostForm
```

Working with Cache files

```
HMTLCacheControl
```

Note:

Loop on the `HTMLEventTask` call till the user indicates by either an `Esc`, `OA-. .`, or `OA-W` key press, or that a Close box has been clicked, that the window should be closed.

The Routines

Housekeeping Routines

HTMLBootInit	Initialises the HTML Tool Set; called only by the Tool Locator must not be called by an application
HTMLStartUp	Starts up the HTML Tool Set for use by an application
HTMLShutDown	Shuts down the HTML Tool Set
HTMLVersion	Returns the version number of the HTML Tool Set
HTMLReset	Resets the HTML Tool Set; called only when the system is reset must not be called by an application
HTMLStatus	Indicates if the HTML Tool Set is active

Global Routines

HTMLInit	Called by an application to prime the HTML Tool Set for use
HTMLParse	Parses raw HTML text into a TERecord
HTMLParse2	Parses raw HTML text into a TERecord with pixel width
HTMLGetError	Returns the last error

Display Routines

HTMLOpenDisplay	Opens an optional display window with a single TextEdit Control
HTMLOpenDisplay2	Opens an optional display window from a supplied window template
HTMLCloseDisplay	Closes the optional display window
HTMLInsertTERecord	Inserts a TERecord into the open display window
HTMLSetTextTERecord	Replaces the entire text of the open display window
HTMLClearDisplay	Clears any text from the open display window
HTMLEventTask	Gives user interaction with the open display window
HTMLEventTask2	Gives user interaction with the open display window
HTMLEventToDisplay	Gives user interaction with the open display from the application's EventRecord
HTMLGetTitle	Returns the page Title
HTMLGotoName	Jumps to a Name Link within the page in the open display window
HTMLGotoName2	Jumps to a Name Link within the page in the open display window
HTMLFindString	Finds and jumps to a string within the open display window
HTMLFindString2	Finds and jumps to a string within the open display window
HTMLGetForm	Returns the Form data from an active form
HTMLGetForm2	Returns the Form data from an active form
HTMLCheckForForm	Returns the number of active Forms in the page of an open display window

Online Routines

HTMLGetPage	Retrieves a web page or file from a supplied URL
HTMLPostForm	Posts a form from the current web page
HTMLSetLogPath	Sets the pathname to be used for the Debug file

Offline Routines

HTMLCacheControl	Saves, Updates, Restores, and Kills page files from the Cache folder
HTMLExtractText	Extracts and prepares text from a TERecord for Printing
HTMLWriteToLog	Allows a client to add custom text to the Debug Log file

\$0182 HTMLBootInit

Initialises the HTML Tool; called only by the Tool Locator.

Warning

An Application must never make this call.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors None

C Call must not be made by an application.

\$0282 HTMLStartUp

Starts up the HTML Tool for use by an application.

Important

Your Application must make this call before it makes any other HTML Tool calls.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors None

C `extern pascal void HTMLStartUp ();`

\$0382 HTMLShutDown

Shuts down the HTML Tool.

Important

If your Application has started up the HTML Tool, the application must make this call before it quits.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors None

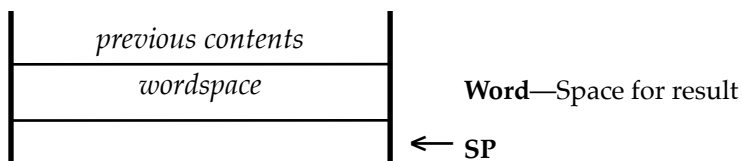
C `extern pascal void HTMLShutDown ();`

\$0482 HTMLVersion

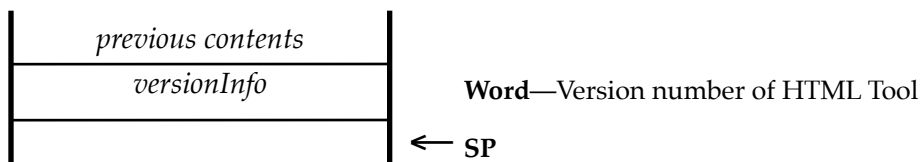
Returns the version number of the HTML Tool.

Parameters

Stack before call



Stack after call



Errors None

C `extern pascal Word HTMLVersion ();`
`Word versionInfo`

\$0582 HTMLReset

Resets the HTML Tool; called only when the system is reset.

Warning

An Application must never make this call.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors None

C Call must not be made by an application.

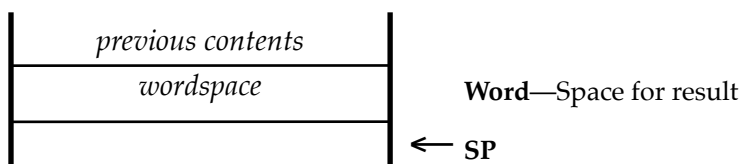
\$0682 HTMLStatus

Indicates whether the HTML Tool Set is active.

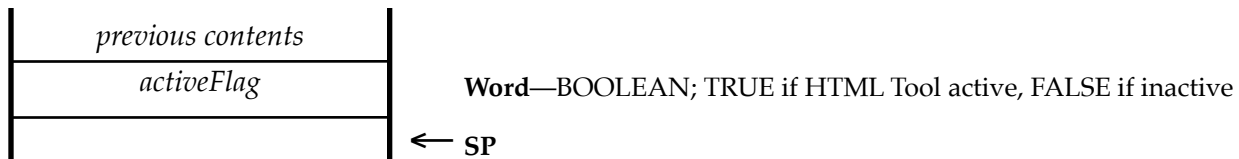
HTMLStatus returns TRUE if HTMLStartup has been called and HTMLShutDown has not been called. The routine returns FALSE if HTMLStartUp has not been called at all or if HTMLShutDown has been called since the last time HTMLStartUp was called.

Parameters

Stack before call



Stack after call



Errors None

C

```
extern pascal Boolean HTMLStatus ();  
Word    activeFlag
```

\$0982 HTMLInit

Prepares the HTML Tool for use by an application.

This call must be made by the application before any of the other calls are made.

The call checks that the required fonts are installed, and any required Tools are currently active. It also checks to see if the optional TalkingTools are present.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors \$82FE htNoTools Required Fonts and Tools not present

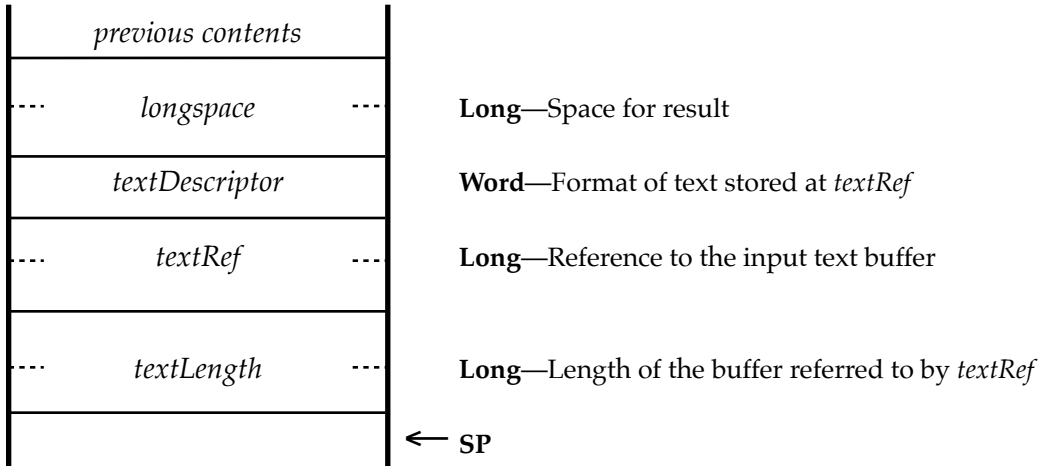
C extern pascal void HTMLInit ()

\$0A82 HTMLParse

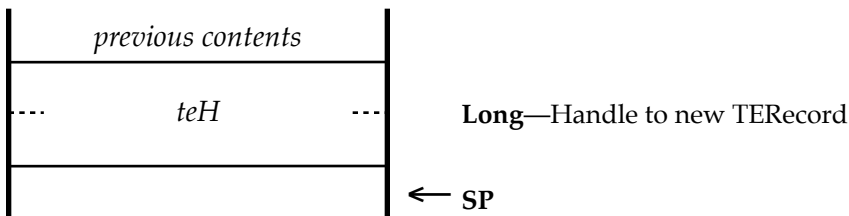
Parses the supplied HTML text into a new TERecord.

Parameters

Stack before call



Stack after call



Errors	\$8202	htBadHandle	A bad Handle was supplied
	\$8206	htNoData	An empty Handle or buffer was supplied
	\$8207	htFailed	The parsing failed
	\$8208	htNotInit	HTMLInit has not been called
	\$82FF	htBadCall	Incorrect parameters supplied

C extern pascal Long HTMLParse (textDescriptor, textRef, textLength);
 Long textRef, textLength, teH
 Word textDescriptor

Note: If a <title> tag was found during parsing, it will be embedded at the start of the TERecord using the invisible font. Unique markers are placed at either end of the string. An application can then easily extract the Title from the page if needed. The Title will never exceed 74 characters, but you need to allow a 78 byte buffer to include the four unique markers: [*A Title Will Be Found In Here*]

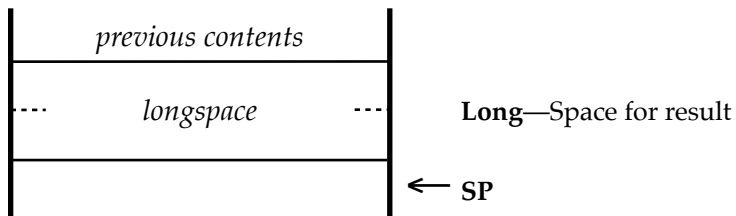
<i>textDescriptor</i>	The format of the text to be parsed, and the type of reference stored in <i>textRef</i> .
Reserved	Bits not used must be set to 0.
<i>dataFormat</i>	bit 0 Defines the format of the passed text: 0 = <i>textRef</i> pointer to a text buffer; <i>textLength</i> contains the length of the buffer (in bytes) 1 = <i>textRef</i> pointer to a Handle holding text; <i>textLength</i> is ignored bit 12 = Set to limit text size to 12pt or larger bit 13 = Display progress thermometer box at position: '3,360,11,510'
<i>textLength</i>	Length of the buffer referenced in <i>textRef</i> . This field is valid only if <i>textRef</i> points to a buffer.

\$0B82 HTMLOpenDisplay

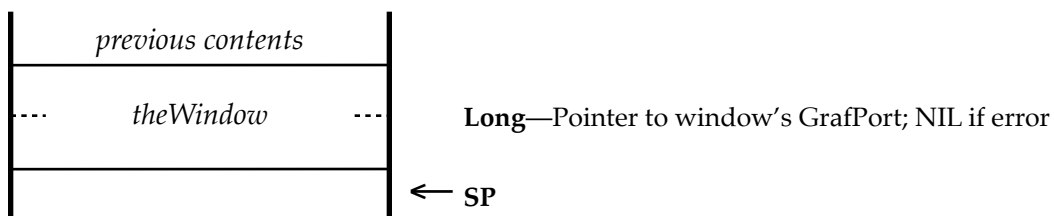
Opens the optional display window. This window is used by many of the calls.

Parameters

Stack before call



Stack after call



Errors	\$8202	htWindowOpen	Window open
	\$8208	htNotInit	HTMLInit has not been called

```

C      extern pascal Long HTMLOpenDisplay ();
        Long      theWindow
  
```

\$0C82 HTMLCloseDisplay

Closes the optional display window.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors

\$8204	htWindowClosed	Window closed
\$8208	htNotInit	HTMLInit has not been called

C

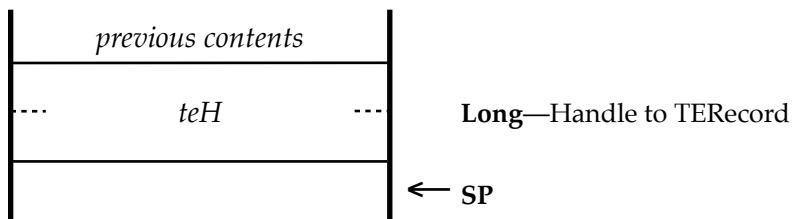
```
extern pascal void HTMLCloseDisplay ();
```

\$0D82 HTMLInsertTERecord

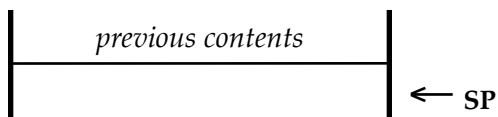
Inserts a TERecord at the end of the text in the optional display window.

Parameters

Stack before call



Stack after call



Errors

\$82-2	htBadHandle	Invalid TERecord
\$8204	htWindowClosed	Window closed
\$8206	htNoData	The TERecord is empty
\$8208	htNotInit	HTMLInit has not been called

C

```
extern pascal void HTMLInsertTERecord (teH);  
Long    teH
```

Note: The calling application must Kill the passed TERecord after the call if it no longer requires it.

\$0E82 HTMLClearDisplay

Clears the text from the optional display window.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors

\$8204	htWindowClosed	Window closed
\$8208	htNotInit	HTMLInit has not been called

C

```
extern pascal void HTMLClearDisplay ();
```

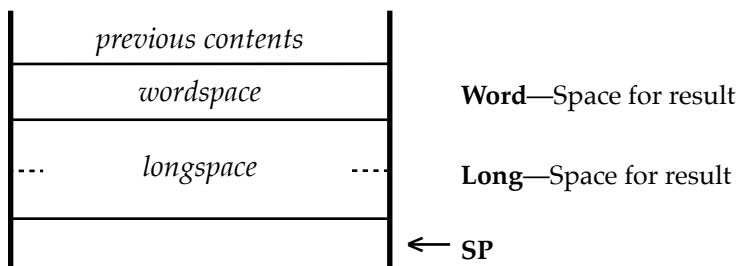
Note: This call creates a new TextEdit control. Your application should call GetCtlHandleFromID to recover the new Handle that has been allocated.

\$0F82 HTMLEventTask

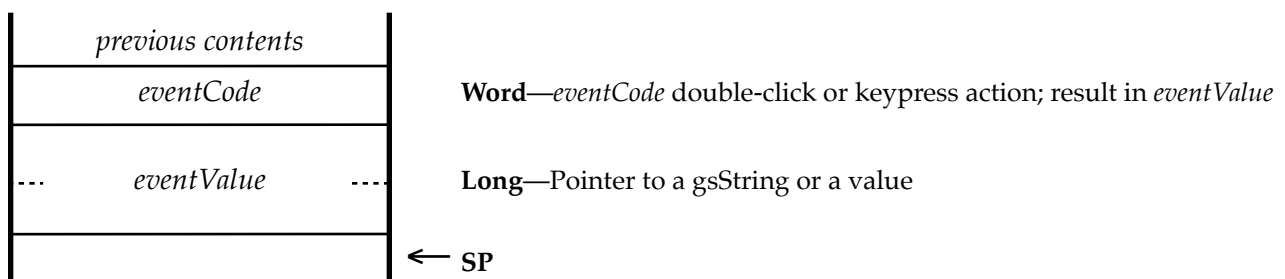
Allows the user to interact with the optional display window.

Parameters

Stack before call



Stack after call



Errors

\$8204	htWindowClosed	Window closed
\$8208	htNotInit	HTMLInit has not been called

C extern pascal Word Long HTMLEventTask ();
 Long eventValue
 Word eventCode

EventRec

eventCode Indicates which item the user double-clicked or which keys they pressed.

- \$00 No item selected
- \$01 URL of a page or email address was double-clicked
- \$02 URL of an image icon was double-clicked
- \$05 a GET submit button was double-clicked
- \$06 a POST submit button was double-clicked
- \$08 a LINKed icon was double-clicked
- \$10 Escape key was pressed
- \$11 OA-. (stop) was pressed
- \$12 OA-W was pressed

eventValue Returns either a Pointer or a value depending on *eventCode*.

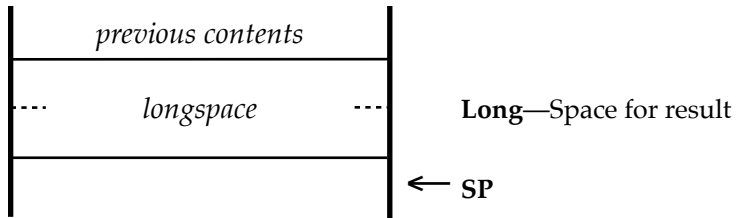
- \$00 Not applicable
- \$01 Pointer to gsString (*wordlength* + *text*)
- \$02 Pointer to gsString (*wordlength* + *text*)
- \$05 LongWord number of a *Form*
- \$06 LongWord number of a *Form*
- \$08 Pointer to gsString (*wordlength* + *text*)
- \$10 Not applicable
- \$11 Not applicable
- \$12 Not applicable

\$1082 HTMLGetTitle

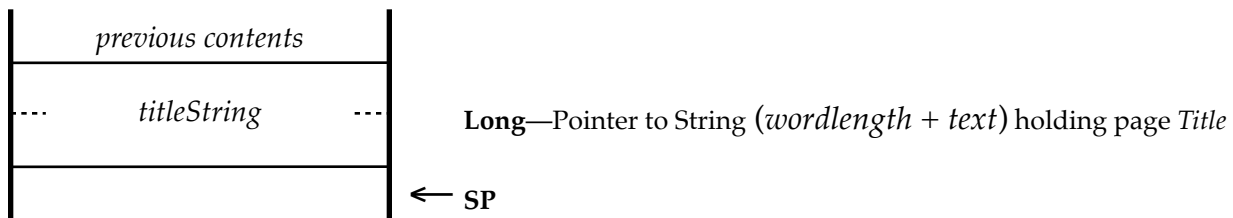
Returns a Pointer to the Title of the page if present. Null if no Title.

Parameters

Stack before call



Stack after call



Errors	\$8204	htWindowClosed	Window closed
	\$8208	htNotInit	HTMLInit has not been called

C

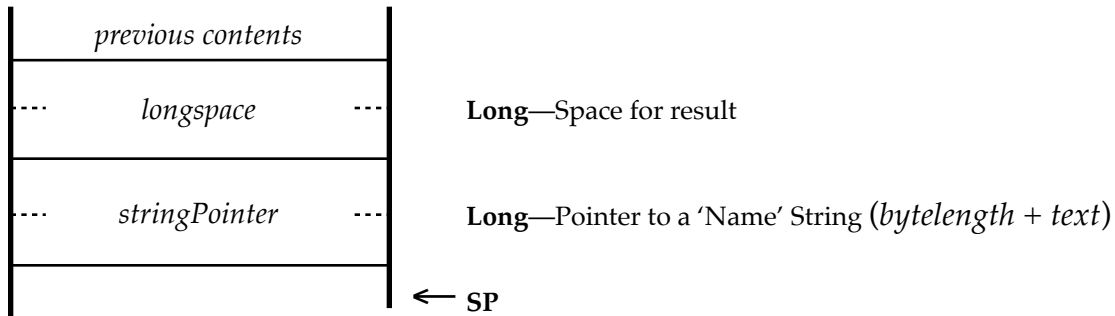
```
extern pascal Long HTMLGetTitle ();  
Long    titleString
```

\$1182 HTMLGoToName

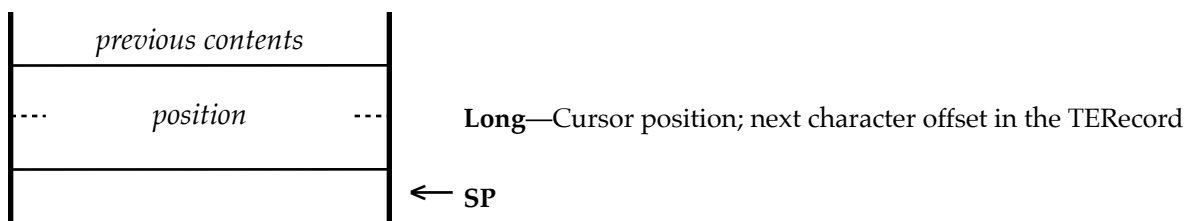
Jumps to a Name Link .

Parameters

Stack before call



Stack after call



Errors	\$8204	htWindowClosed	Window closed
	\$8207	htFailed	'Name' String not found
	\$8208	htNotInit	HTMLInit has not been called
	\$82FF	htBadCall	Incorrect parameters supplied

C

```
extern pascal Long HTMLGoToName (stringPointer);  
Long    position, stringPointer
```

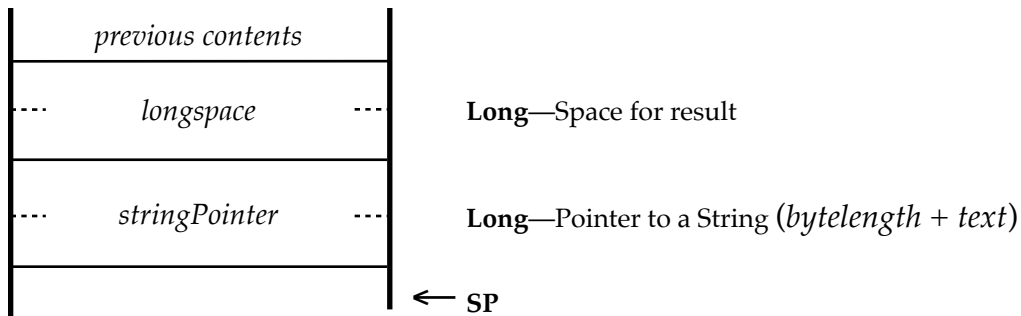
Note: If the target 'Name' is found, the cursor selection will be placed after the 'Name', and the text will be scrolled to place that line at the top of the screen. If you need to put the line elsewhere on the screen, use the returned Cursor position to calculate the scrolling offset.

\$1282 HTMLFindString

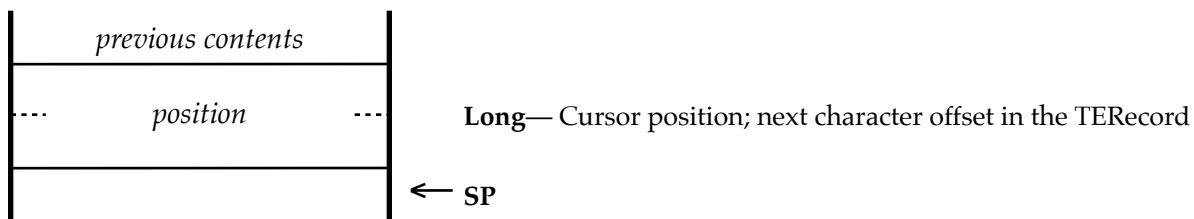
Finds the next occurrence of 'String', searching from the current cursor position.

Parameters

Stack before call



Stack after call



Errors	\$8204	htWindowClosed	Window closed
	\$8207	htFailed	'Name' String not found
	\$8208	htNotInit	HTMLInit has not been called
	\$82FF	htBadCall	Incorrect parameters supplied

C

```
extern pascal Long HTMLFindString (stringPointer);  
Long      position, stringPointer
```

This call only searches the main TextEdit control (ID \$7000), and will ignore hidden text using the SIS-4 font. Please refer to the HTMLFindString2 call for more flexible searching.

Note: If the target string is found, the text will be selected, and scrolled to place it in the middle of the screen.

\$1382 HTMLGetForm

Returns Form data.

Parameters

Stack before call

<i>previous contents</i>	
<i>wordspace</i>	Word —Space for result
... <i>longspace</i> ...	Long —Space for result
... <i>longspace</i> ...	Long —Space for result
<i>formNumber</i>	Word —Number of Form
	← SP

Stack after call

<i>previous contents</i>	
<i>postCode</i>	Word — <i>postCode</i> 'G' or 'P' for 'GET' or 'POST'
... <i>urlPointer</i> ...	Long —Pointer to TEREcord holding URL
... <i>formDataPointer</i> ...	Long —Pointer to TEREcord holding Form Data
	← SP

Errors	\$8204	htWindowClosed	Window closed
	\$8207	htFailed	Form does not exist
	\$8208	htNotInit	HTMLInit has not been called
	\$82FF	htBadCall	Incorrect parameters supplied

C

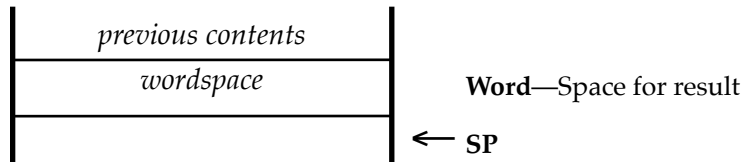
```
extern pascal Word Long Long HTMLGetForm (formNumber);
Long      urlPointer, formDataPointer
Word      formNumber, postCode
```

\$1482 HTMLCheckForForm

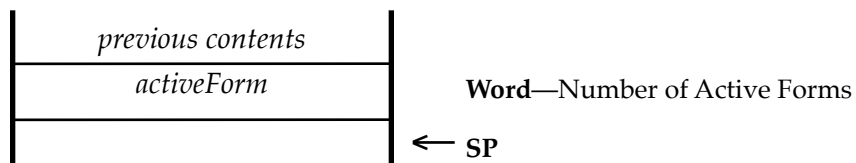
Returns the number of Forms within the page.

Parameters

Stack before call



Stack after call



Errors None

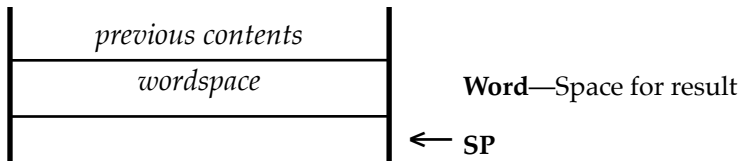
C `extern pascal Word HTMLCheckForForm ();`
 `Word activeForm`

\$1582 HTMLGetError

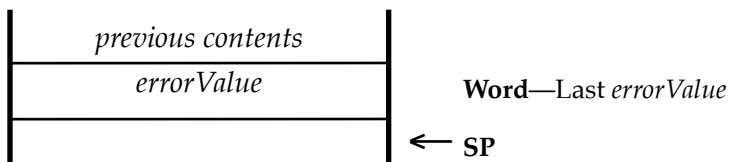
Returns the last *errorValue*.

Parameters

Stack before call



Stack after call



Errors None

C extern pascal Word HTMLGetError ();
 Word errorValue

\$1682 HTMLParse2

Parses the supplied HTML text into a new TEREcord.

Parameters

Stack before call

<i>previous contents</i>	
--- <i>longspace</i> ---	Long —Space for result
<i>textDescriptor</i>	Word —Format of text stored at <i>textRef</i>
--- <i>textRef</i> ---	Long —Reference to the input text buffer
--- <i>textLength</i> ---	Long —Length of the buffer referred to by <i>textRef</i>
<i>rightMargin</i>	Word —The right margin of the TextEdit control in pixels
	← SP

Stack after call

<i>previous contents</i>	
--- <i>teH</i> ---	Long —Handle to new TEREcord
	← SP

Errors	\$8202	htBadHandle	A bad Handle was supplied
	\$8206	htNoData	An empty Handle or buffer was supplied
	\$8207	htFailed	The parsing failed
	\$8208	htNotInit	HTMLInit has not been called
	\$82FF	htBadCall	Incorrect parameters supplied

C extern pascal Long HTMLParse (textDescriptor, textRef, textLength, rightMargin);
Long textRef, textLength, teH
Word textDescriptor, rightMargin

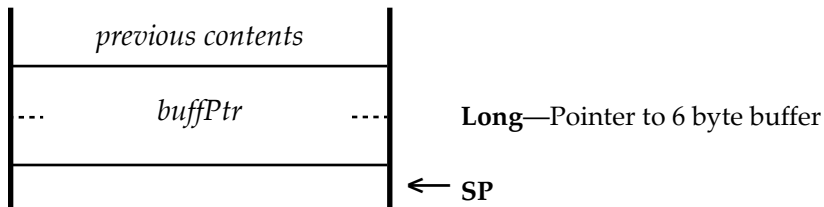
Note: This is identical to the HTMLParse call, but with the addition of the *rightMargin* value being passed. This value controls how <center>, <hr align=center> and <hr align=right> tags are handled. You should pass the usable width in pixels for the TextEdit control that is being used.

\$1782 HTMLEventTask2

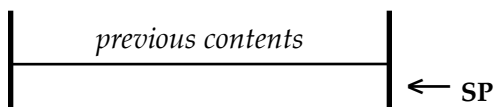
Allows the user to interact with the optional display window.

Parameters

Stack before call



Stack after call



Errors	\$8204	htWindowClosed	Window closed
	\$8208	htNotInit	HTMLInit has not been called

C

```
extern pascal void HTMLEventTask2 (buffPtr);  
Long      buffPtr
```

This call returns the same information as HTMLEventTask, but with the results in a data buffer pointed at by *buffPtr*, rather than on the stack.

On output *eventCode* and *eventValue* hold the returned values. Refer to Page 17-18 for more details.

buffer 6 bytes:

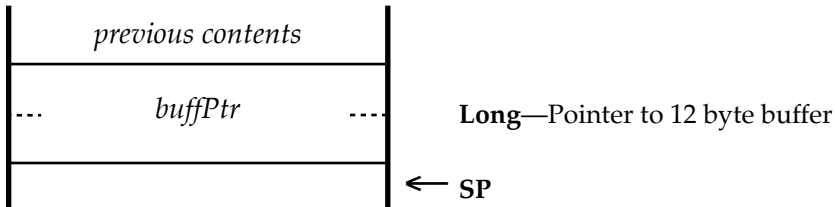
eventCode two bytes
eventValue four bytes

\$1882 HTMLGetForm2

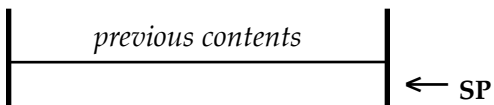
Returns Form data.

Parameters

Stack before call



Stack after call



Errors	\$8204	htWindowClosed	Window closed
	\$8207	htFailed	Form does not exist
	\$8208	htNotInit	HTMLInit has not been called
	\$82FF	htBadCall	Incorrect parameters supplied

C

```
extern pascal void HTMLGetForm2 (buffPtr);  
Long buffPtr
```

This call returns the same information as GetForm, but with the results in a data buffer pointed at by *buffPtr*, rather than on the stack.

On input, *formNumber* holds the number of the form to retrieve the data from, and on output *postCode*, *urlPointer*, and *formDataPointer* hold the returned values.

buffer 12 bytes:

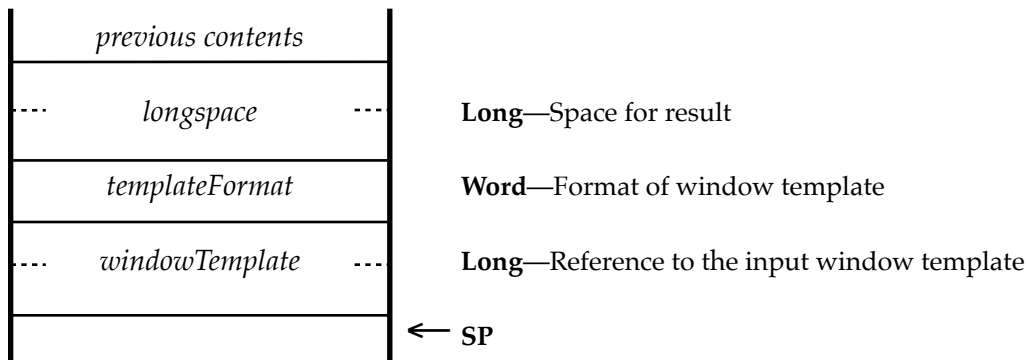
<i>formNumber</i>	two bytes	<i>Number of Form (input)</i>
<i>postCode</i>	two bytes	<i>'G' or 'P' for 'GET' or 'POST' (output)</i>
<i>urlPointer</i>	four bytes	<i>Pointer to TERecord holding URL (output)</i>
<i>formDataPointer</i>	four bytes	<i>Pointer to TERecord holding Form Data (output)</i>

\$1982 HTMLOpenDisplay2

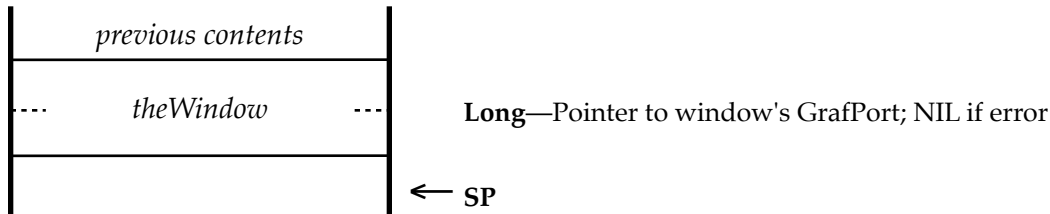
Opens a custom display window from a supplied template using *NewWindow2*.

Parameters

Stack before call



Stack after call



Errors

\$8202	htWindowOpen	Window open
\$8208	htNotInit	HTMLInit has not been called

C

```
extern pascal Long HTMLOpenDisplay2 (templateFormat,
windowTemplate);
Long      windowTemplate, theWindow
Word      templateFormat
```

templateFormat Indicates the format of the windowTemplate reference:

\$00	Pointer
\$01	Handle
\$02	Resource

This call allows a custom window template to be used, rather than the basic window opened by HTMLOpenDisplay. The window should then be interacted with in the same way.

Note: The window template must include a TextEdit control with these settings:

```
TextEdit control with the ID $7000
moreFlags    = $7400      textflags    = $44000000
```

Note: Optionally you can have a Stop Icon or Button control with the ID \$8000:

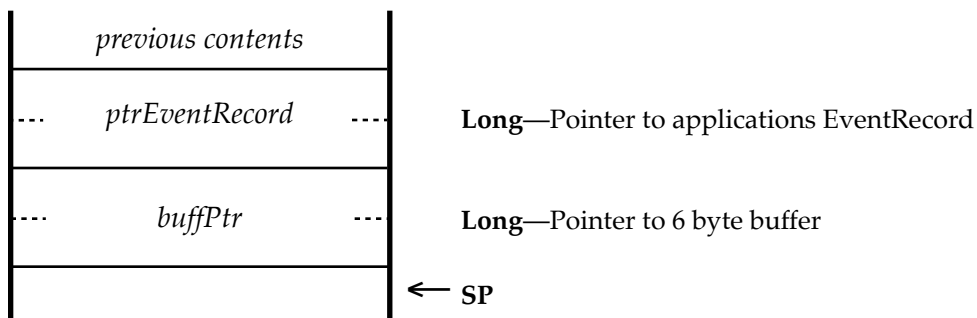
```
key equivalent = Escape key,
```

\$1A82 HTMLEventToDisplay

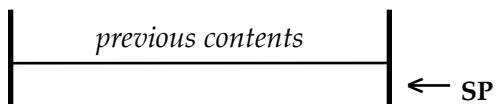
Allows the user to interact with the optional display window.

Parameters

Stack before call



Stack after call



Errors	\$8204	htWindowClosed	Window closed
	\$8208	htNotInit	HTMLInit has not been called

C

```
extern pascal void HTMLEventToDisplay (ptrEventRecord, buffPtr);
Long ptrEventRecord, buffPtr
```

This call uses the *EventRecord* from the calling application, then returns the same information as HTMLEventTask, with the results returned in a data buffer pointed at by *buffPtr*.

On output *eventCode* and *eventValue* hold the returned values. Refer to Pages 17-18 for more details.

buffer 6 bytes:

eventCode two bytes
eventValue four bytes

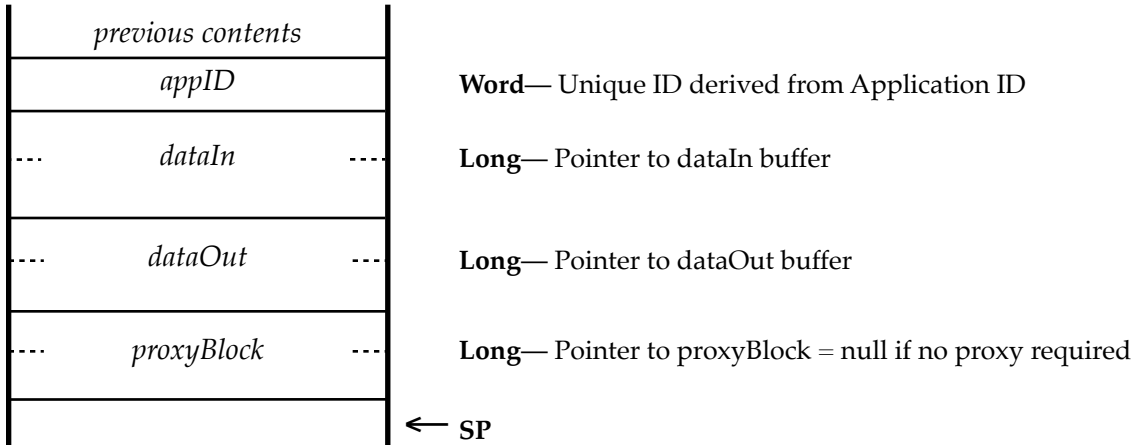
HTMLEventTask works with windows opened using HTMLOpenDisplay in a closed loop. For an application to interact with the window externally, use HTMLEventToDisplay instead. This allows events to be sent directly to the window from the application. Refer to Page 17 for a description of the returned *eventCode* and *eventValue* values.

\$1B82 HTMLGetPage

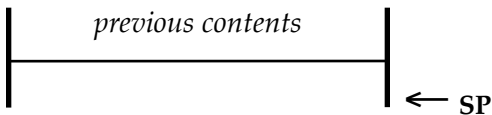
Retrieves a file from a web server, and optionally parses it into a TextHandle.

Parameters

Stack before call



Stack after call



Errors	\$8202	htBadHandle	The passed Handle is bad
	\$8205	htBadParms	Passed data is bad
	\$8208	htNotInit	HTML Tool not initialised
	\$8209	htNoTCPIP	TCP/IP is not installed or not active
	\$820C	htUserKilled	The User stopped the process

C

```
extern pascal void HTMLGetPage (appID, dataIn, dataOut);
Long      dataIn, dataOut
Word      appID
```

This call retrieves the file pointed to by the passed URL, and returns the data in a Handle. The returned data would normally be a web page, but it can also be any file that is held on a web server. If the URL is prefixed with `file://` it will be loaded from a local disk instead.

Optionally, if the tag `<html>` is found in the returned data, the call can continue and call `HTMLParse` to return a fully formatted TextEdit Handle. The returned Text Edit Handle then only needs to be inserted into the screen display using `HTMLInsertTERecord`.

With just two Tool calls, `HTMLGetPage` and `HTMLInsertTERecord`, you can display a formatted web page in a web browser, then with `HTMLEventTask` or `HTMLEventToDisplay`, you can interact with that display. Please refer to Pages 17-18 and 29 for more details.

dataIn buffer 12, 16 or 22 bytes:

<i>URLHandle</i>	four bytes	<i>Handle holding the target URL</i>
<i>webPort</i>	two bytes	<i>The target port. A value of \$0000 = web port \$80</i>
<i>inputSettings</i>	two bytes	<i>Bit 0 = GET return file data in a Handle Bit 1 = GET return and process into a TextEditHandle Bit 2 = HEAD returns the full header data in a Handle Bit 3 = Browser supports Cookies (See Cookies page) Bit 4 = Return Port & IPAddress from DNS server Bit 5 = Use the Port & IPAddress from DNS server Bit 6 = Save Page data to supplied PathName Bit 7 = Use a Client Thermometer call Bit 12 = Forces 12pt minimum sized font Bit 13 = Progress thermometer box at: '3,360,11,510' Bit 14 = Save debug payload data Bit 15 = Save debug header data</i>
<i>dataFolder</i>	four bytes	<i>A pointer to GS/OS pathname of a data folder for holding app specific data, or to save page data to (Bit 6)</i>
<i>cookieList</i>	four bytes	<i>Handle holding list of Cookies to send (Optional)</i>
<i>usePortIPAddress</i>	six bytes	<i>Use this Port & IPAddress</i>
<i>thermCallPointer</i>	four bytes	<i>Pointer to a Thermometer data block</i>

dataOut buffer 36, 40, or 46 bytes:

<i>errorFlag</i>	two bytes	<i>Refer to Page 34</i>
<i>dataFormat</i>	two bytes	<i>Bit 0 = Handle returned Bit 1 = TEHandle returned Bit 2 = File was loaded from disk Bit 3 = Page data was saved as a file to disk Bit 14 = No Cache was reported Bit 15 = <html> found</i>
<i>dataHandle</i>	four bytes	<i>Returned Handle or TextEdit Handle</i>
<i>Content-Length</i>	four bytes	<i>Size of payload data returned from server</i>
<i>Content-Type</i>	four bytes	<i>Pointer to pString returned from the header data</i>
<i>Date</i>	four bytes	<i>Pointer to pString returned from the header data</i>
<i>Last-Modified</i>	four bytes	<i>Pointer to pString returned from the header data</i>
<i>Moved-URL</i>	four bytes	<i>Handle of moved URL (see Page 37)</i>
<i>Extras</i>	four bytes	<i>Not used for HTMLGetPage</i>
<i>Last-SearchPage-URL</i>	four bytes	<i>Not used for HTMLGetPage</i>
<i>cookieList</i>	four bytes	<i>Handle holding list of returned Cookies (Optional)</i>
<i>returnPortIPAddress</i>	six bytes	<i>Returned Port & IPAddress from DNS server</i>

Note: Returned Handles must be disposed of by the calling application.

HTMLGetPage is a powerful call that allows the HTML Tool to be the core engine of a web browser. When a URL is passed to the call, and Bit 1 of *inputSettings* is set, if an <html> tag is found in the returned data, an HTML formatted Text Edit Handle will be returned. That data can then be inserted into the TE Control of a display, and can be interacted with to display further pages, or post the results of any forms on a page. See Pages 17-18 for more details.

If you set Bit 2 of *inputSettings*, the call sends a HEAD command instead of a GET command. This returns the size, type, and other information from the file, as well as the full header in a Handle, but not the payload data itself. This is useful if you wish to download files from a web server, rather than retrieve a web page, as you can first determine if a particular file is present, and what is size it is. Note that “*Chunked*” transfers will not return a size.

In most cases, you will want to follow the HEAD call with a full GET call for the same URL. This would normally involve calling the DNS server twice for the same URL. When making the HEAD call, if you set Bit 4 of *inputSettings*, the Port and IPAddress returned from the DNS server will be returned in the six *returnPortIPAddress* bytes. If you then set Bit 5 of *inputSettings*, and pass these same values back in *usePortIPAddress*, these values will be used instead of making a second DNS call with the passed URL. You cannot pass a Port and IPAddress if you are making a HEAD call.

Note that the *dataIn* and *dataOut* buffers are larger if set Bits 4 & 5, so make sure your web browser client has allowed for that, and also be careful when using this feature that you do not pass back invalid values, or values that do not relate to the URL you are calling.

Set Bit 13 of *inputSettings* to display an optional progress thermometer in the Menu bar. The thermometer will be displayed at position '3,360,11,510', so if you use it, make sure you do not have any menus displaying in that position.

Note: If the passed URL reports a “300 Moved Permanently” or “302 Moved Temporarily” Error, the new URL Location will be returned in the *Moved-URL* Handle.

Proxy Connections

If you use a proxy connection to connect to a web server, pass a pointer in *proxyBlock* to a formatted Block holding the URL of your proxy server, and optionally include an alternative port number, a username, and a password.

Note: If you do not require to use a proxy connection, pass a value of zero for the pointer.

If a pointer is passed, the *proxyBlock* must contain 773 bytes, even if the pStrings are empty:

ProxyBlock:

Proxy URL (257 bytes) pString holding the server URL
Proxy Port (word) passa a value of zero for port = 80
Proxy Username (257 bytes) pString holding the userName
Proxy Password (257 bytes) pString holding the password

The Proxy URL must be provided.

The Proxy Port is optional, and if a value of zero is passed, the default Port 80 will be used.

The Proxy Username is optional, pass a zero length pString if it is not required.

The Proxy Password is optional, pass a zero length pString if it is not required.

If a Proxy connection has been requested, and the server requires Authentication, and a user/pass has not been passed or is incorrect, a popup window will open for them to be entered. If no user/pass was passed, the HTMLTool will remember the new correct user/pass until the IIgs is rebooted. It would be up to the user to correct the entry in the calling application so that did not happen again.

Page Authentication

Some web pages may require authentication. In such a case, a popup window opens, with the Realm identifier as title. Check the “Remember Me” box for the user/pass to be saved to a “Passwords” file in the *dataFolder* of the application. The next time that page, or a sub folder is accessed, the saved user/pass will be used to access it. If you do not check “Remember Me”, the user/pass will only be remembered for the current session.

HTTPS URLs

HTTPS secure connections are not supported by the TCP/IP stack, so if a URL is passed that parses out to an HTTPS address, an *htBadPath* error will be returned.

The application should wherever possible first check it is not passing HTTPS URLs for either the HTMLGetPage or HTMLPostForm calls, and then respond suitably if an *htBadPath* error is reported.

If you are using a Proxy connection, some Proxy servers may allow an HTTPS URL to be passed. In this case, the Tool will not trap HTTPS URLs, but if the Proxy server cannot handle it, may return an error if one is passed.

Debug Logs

Optionally you can turn on a Debug log file by setting bits 15 & 14 of *inputSettings*. Unless you change the default path using the HTMLSetLogPath call, the Debug file will be saved in the root directory of your Boot disc, with the name "HTTP.Log".

Set Bit 15 to save the connection handshake headers, and Bit 14 to save the payload data. Set both Bits 14 & 15 to save both the headers and payload to the debug file.

Note, if you are making an HTMLGetPage HEAD call before a full HTMLGetPage call, you may wish to clear bits 15 & 14, to avoid seeing the results of the HEAD call in the Debug log.

Save to PathName

If Bit 6 of *inputSettings* is set, a pointer to a PathName supplied in *dataFolder* will be used to create a file to save the incoming page data to, and no data Handle will be returned.

The file will be saved as a data file (FileType \$06), and if the file pointed to in PathName already exists, it will be deleted.

This call is useful if you wish to download a file directly from a web server, as a file of any size can be downloaded directly to disk without the data being kept in memory in a Handle during the call.

Client Thermometer

As an alternative to, or in addition to the small menu bar thermometer provided by the HTMLGetPage call, set Bit 7 of *inputSettings*, and provide a Pointer in *thermCallPointer* to a Data Block in your app to control a Thermometer Control within the app itself:

Thermometer Data Block:

```
ThermCtl      dc i4'0'           ; Pointer to an update routine in the app
ThermScale    dc i2'$015F'       ; thermometer scale
ThermPosition ds 2               ; position passed back in call from Tool
```

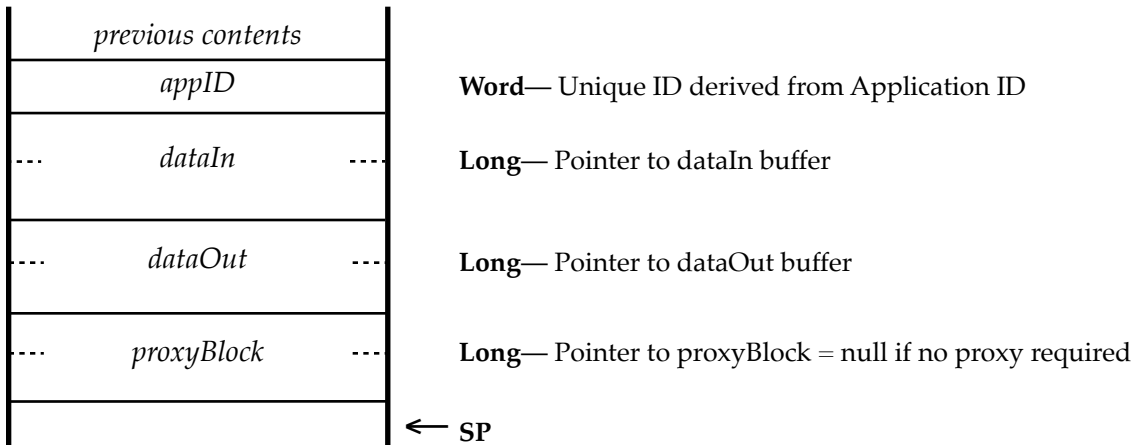
The Control must already exist, and the update routine should only set the Control's value using the value passed back in *ThermPosition*.

\$1C82 HTMLPostForm

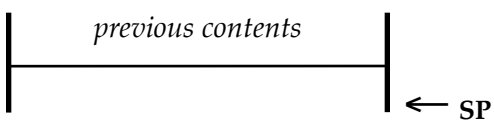
Posts Form data to the web server.

Parameters

Stack before call



Stack after call



Errors	\$8202	htBadHandle	The passed Handle is bad
	\$8205	htBadParms	Passed data is bad
	\$8208	htNotInit	HTML Tool not initialised
	\$8209	htNoTCPIP	TCP/IP is not installed or not active
	\$820B	htGenError	GS/OS and general errors
	\$820C	htUserKilled	The User stopped the process

C

```
extern pascal void HTMLPostForm (appID, buffPtr);  
Long      buffPtr  
Word      appID
```

This call Posts the data returned when you double-click a submit button within a Form, and sends it to the target web server. Depending on how the Form was defined, the call will use either a GET or POST command to send the data.

You can either retrieve the Form number using HTMLEventTask, HTMLEventTask2, or HTMLEventToDisplay and and pass this number to the call, and let HTMLPostForm handle the entire process, or alternatively you can manually use the Form number to first retrieve the two TEHandles using HTMLGetForm or HTMLGetForm2, and pass that data instead.

Note: If a path to a local file on disk is specified for the target URL, the call will fail.

The two TEHandles hold the target URL, and the payload data for the Form. Internally, the HTMLPostForm call uses that data to build the composite URL for a GET command, or the payload Data for a POST command. Refer to the examples below.

dataIn buffer 24, 28 or 34 bytes:

<i>pageURL</i>	four bytes	<i>Handle holding the target URL</i>
<i>formPort</i>	two bytes	<i>The target port. A value of \$0000 = web port \$80</i>
<i>formPrefs</i>	two bytes	<i>Bit 0 = GET return file data in a Handle Bit 1 = GET return and process into a TextEditHandle Bit 3 = Browser supports Cookies (See Cookies page) Bit 4 = Return Port & IPAddress from DNS server Bit 5 = Use the Port & IPAddress from DNS server Bit 12 = Forces 12pt minimum sized font Bit 13 = Progress thermometer box at: '3,360,11,510' Bit 14 = Save debug headers Bit 15 = Save debug headers</i>
<i>formNumber</i>	two bytes	<i>Form Number (if zero, the following three parms must be set)</i>
<i>postCode</i>	two bytes	<i>'G' or 'P' for GET or POST</i>
<i>formHandle1</i>	four bytes	<i>Pointer to FormURL in TEHandle (Handle will be disposed of)</i>
<i>formHandle2</i>	four bytes	<i>Pointer to FormData in TEHandle (Handle will be disposed of)</i>
<i>dataFolder</i>	four bytes	<i>A pointer to GS/OS pathname of a data folder for holding app specific data</i>
<i>cookieList</i>	four bytes	<i>Handle holding list of Cookies to send (Optional)</i>
<i>usePortAddress</i>	six bytes	<i>Use this Port & IPAddress</i>

On entry, you must set *formPort*, *formPrefs*, and the *pageURL*, then either *formNumber*, or *postCode*, *formHandle1*, *formHandle2* and *dataFolder*.

dataOut buffer 36, 40 or 44 bytes:

<i>errorFlag</i>	two bytes	<i>See below</i>
<i>dataFormat</i>	two bytes	<i>Handle, TEHandle, No Cache, <html> found (see Page 31)</i>
<i>dataHandle</i>	four bytes	<i>Returned Handle or TextEdit Handle</i>
<i>Content-Length</i>	four bytes	<i>Pointer to pString returned from the header data</i>
<i>Content-Type</i>	four bytes	<i>Pointer to pString returned from the header data</i>
<i>Date</i>	four bytes	<i>Pointer to pString returned from the header data</i>
<i>Last-Modified</i>	four bytes	<i>Pointer to pString returned from the header data</i>
<i>Moved-URL</i>	four bytes	<i>Pointer to Handle of moved URL (see Page 37)</i>
<i>Extras</i>	four bytes	<i>loWord 'P' or 'G' depending on how call was POSTed hiWord the number of the Form that was POSTed</i>
<i>Last-SearchPage-URL</i>	four bytes	<i>Pointer to GS/OS String of last page and search Payload</i>
<i>cookieList</i>	four bytes	<i>Handle holding list of returned Cookies (Optional)</i>
<i>returnPortIPAddress</i>	six bytes	<i>Returned Port & IPAddress from DNS server</i>

Errors returned in *errorFlag* if an error on the stack is *htGenError*:

\$0001 = *Bad Connection*

\$0002 = *No response*

\$0004 = *Bad response*

\$0008 = *GS/OS error file error or would not open*

\$0010 = *GS/OS write error*

\$0020 = *GS/OS volume full*

\$0040 = *TCP/IP DNR abort error*

\$0xxx = *HTTP error (lower 12 bits hold a value such as a '404' page not found error)*

\$8000 = *User killed/User stopped*

The calling application must dispose of the *dataHandle* Handle when it has finished with it.

Note: A Handle with the URL of the current page should be passed in *pageURL*. If a URL was not found within the Form data, this URL will be used to POST the data. If you know that a valid URL is being supplied in *formHandle1*, then you need not also pass it in *pageURL*.

Note: Any payload data will be returned in a Handle, or optionally if it contains HTML code, be processed into a TEHandle. Display a TEHandle in the main TextEdit control of your display window using *HTMLInsertTERecord*, or if not already processed, you can first process a Handle of HTML data using *HTMLParse* before calling *HTMLInsertTERecord*.

Method One

The simplest way to use the *HTMLPostForm* call is to copy the Form number returned from *HTMLEventTask*, *HTMLEventTask2*, or *HTMLEventToDisplay* into *formNumber*, and add the page URL. A GET or POST call will then be automatically made.

On return, *Extras* will hold either a 'G' or 'P', depending on the actual call that was made.

The *dataHandle* will hold either a raw data Handle, or the processed TEHandle of the page depending on what you asked for in Bits 0 and 1 of *formPrefs*, and what was returned.

As a check for an <html> tag is always made before any automatic processing of the data into a TEHandle, *dataFormat* will tell you what kind of Handle was returned, regardless of what you had asked for in *formPrefs*.

Method Two

This method gives you considerably more control of the form Posting process. In this case, *formNumber* must be set to zero. You must set *postCode* with either a 'G' or a 'P' for Get and POST, and must supply the required URL, and the form data in the two TEHandles, *formHandle1* and *formHandle2*.

TEHandle *formHandle1* holds the target URL, and TEHandle *formHandle2* the actual Form data. This data can be retrieved using the Form number and *HTMLGetForm*, or *HTMLGetForm2*, or as you have control of both of these two TEHandles, if necessary, you can change or build the data in *formHandle2* before making the call.

If the call is a GET call, then *HTMLPostForm* will construct the URL correctly from the two TEHandles. For a POST call, *formHandle2* will be sent as the payload.

As with Method One, any returned page will be returned in *dataHandle* and the type of Handle in *dataFormat*.

Moved-URL - Moved Permanently Error

If the passed *pageURL* results in a “301 Moved Permanently” or “302 Moved Temporarily” response, the new *Moved-URL* is returned to the HTMLGetPage and HTMLPostForm calls.

It is up to the calling application to check for a *Moved-URL* from either of the calls.

If a *Moved-URL* is returned, the application can retrieve the new URL from the Handle. The application can then offer the new URL to be passed once again to either of the calls.

Proxy Connections and Page Authentication

Refer to Page 32 for further details.

Using the returned DNS server IPAddress

In most cases, you will want to follow a HEAD call with a full GET or POST call for the same URL. This would normally involve calling the DNS server twice for the same URL. When making the HEAD call, if you set Bit 4 of *inputSettings*, the Port and IPAddress returned from the DNS server will be returned in the six *returnPortIPAddress* bytes. If you then set Bit 5 of *inputSettings*, and pass these same values back in *usePortIPAddress*, these values will be used instead of making a second DNS call with the passed URL. You cannot pass a Port and IPAddress if you are making a HEAD call.

For a POST call, it will probably be less useful to have it return the *returnPortIPAddress* Port & IPAddress, but if you have made a HEAD call before the POST call, then you can use the results from that with the POST call, to save the DNS server from being called once more.

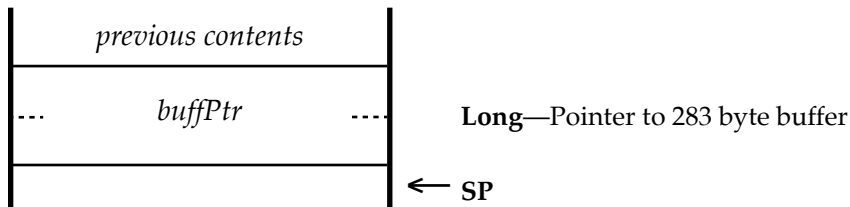
Note that the *dataIn* and *dataOut* buffers are larger if set Bits 4 & 5, so make sure your web browser client has allowed for that, and also be careful when using this feature that you do not pass back invalid values, or values that do not relate to the URL you are calling.

\$1D82 HTMLCacheControl

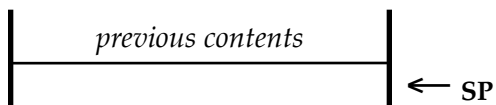
Allows the user to save and recover pages complete with Form data.

Parameters

Stack before call



Stack after call



Errors	\$8204	htWindowClosed	Window closed
	\$8205	htBadParms	Bad values supplied
	\$8208	htNotInit	HTMLInit has not been called
	\$820A	htBadPath	Prefix 8 invalid

C extern pascal void HTMLCacheControl (buffPtr);
Long buffPtr

buffer 283 bytes:

<i>resultCode</i>	word	If full Save/Restore/Update = \$8000
<i>taskCode</i>	word	required task
<i>cacheNumber</i>	thirteen bytes	name of cache file: 'C123456789123'
<i>cacheFlags</i>	word	Bit 11 = Save full pages (see No Cache section) Bit 12 = Page returned from Post call Bit 13 = Server requested password Bit 14 = Server requested No cache Bit 15 = Short cache application request
<i>userFlags</i>	long	optional flags for use by application
<i>pageLength</i>	long	Size of payload data returned from server
<i>cacheURL</i>	256 bytes	used for Save/Restore

Note: Before entry to the Cache call, the application must set Prefix 7 to point to the folder that will hold the Cache files.

The HTMLCacheControl call handles the saving of the *cacheURL* along with various flags and values, and the complete text and style content of the page, along with any Form data from the page into a full cache file. Optionally, you can save just the *cacheURL* and flags as a short cache file. This call enables a cache and navigation strategy for your application, and aids restoring pages from a navigation stack, along with changes that may have been made to any embedded Forms, without needing to retrieve the page again from the web server.

The filenames used for the files in the Cache folder are unique, and must not be changed by the application. They are constructed from the current date and time, so there will never be a duplicate file of the same name created. They have a 'C' at the start to allow the unique numeric name to be accepted by GS/OS.

taskCode: \$01 = Save current page to cache folder
 \$02 = Update current page in cache folder (updates data only)
 \$03 = Restore page from cache folder
 \$04 = Remove page from cache folder
 \$05 = Remove All pages from cache folder
 \$06 = Clean No Cache pages (see No Cache section)
 set bit 15 to save page to passed cacheNumber (saves as new page)

Working with the Cache call

When you have downloaded, Parsed, and Inserted your page into the TextEdit control of the optional or custom window, call HTMLCacheControl to Save the current page to a Cache file, so you can quickly recall the page later. Make a note of the *cacheNumber* that was returned in your History or Navigation list, and when you wish to show the page again, use that *cacheNumber* to Restore the page. This will allow you to quickly display a page from the Cache without having to download and Parse the page once more.

When you move away from that page, either by calling up a new URL, or when your application Quits, use the Update call to keep an updated copy of the page using the same *cacheNumber*. This will preserve any changes you may have made to a Form entry.

If you remove an entry from your History list, call the Remove Page call using the *cacheNumber* to clear the related file from the cache folder. If you wish to empty the entire folder, use the Remove All call.

Note: If a page was marked as “No Cache” by the server, you should respect that by setting Bit 14 of the *cacheFlags* when you Save or Update the page. Refer to the No Cache section.

Save Current Page = 1

Saves the *cacheURL* and *userFlags*, text, style, and Form data from a page to a cache file. The filename is built using a unique number, and returns that unique number for the file in *cacheNumber*. Make a note of that number in your navigation stack for later recall.

If Bit 15 of *cacheFlags* is set, only the *cacheURL* and *Flags* will be saved. This allows for shorter cache files, but when Restored, the page will need to be retrieved again from the server.

If Bit 15 of *taskCode* is set, you must supply the *cacheNumber* on entry. The data will then be saved to that file. This allows you to preserve your navigation history stack, and save any changes back to the same cache file.

Update Current Page = 2

Updates the Form data from a page, using the number passed in *cacheNumber*. This will update the *cacheURL*, *userFlags* and Form data. If a short cache had been saved, only the *cacheFlags* will be updated. If it had been a full cache file, and is now Updated as a short cache file, the page and form data will be removed from the cache file.

Restore Page = 3

Loads the page from disk using the number passed in *cacheNumber*, and replaces the current display with the saved page data. Any Form data will be restored. If a short cache had been saved, the page itself will need to be retrieved, parsed, and redrawn again.

Remove Page = 4

Deletes the saved page from the Cache folder using the passed *cacheNumber*.

Remove All Pages = 5

Deletes all the Cache files from the Cache folder.

Clean No Cache Pages = 6

If you have chosen to save full pages from those marked as “No cache” by the server, you must call this at Quit to strip the text and form data from any marked “No Cache” pages.

User Flags

The four byte long word *userFlags* is saved and restored to and from the cache file. The calling application can use these bytes as they wish, perhaps to store data or flags related to the page, such as the cursor position.

No Cache Page Handling

Some servers may request that you do Not Cache their pages, and it is up to the browser to respect this request by first checking Bit 14 of the returned *cacheFlags*, and then setting Bit 14 again in the *cacheFlags* whenever you Save or Update that page. However, if the IIS is not accelerated, this can cause some long delays for a page to be downloaded again as you go back and forwards through your navigation stack.

To allow a browser to behave more responsively for No Cache pages, set Bit 11 as well as Bit 14 of the *cacheFlags* when you Save or Update a requested No Cache page, and they will then be saved as full cache pages to the cache files instead of short cache pages.

If you have saved any No Cache pages as full cache files, it is important that when your browser Quits, you call `HTMLCacheControl` with *taskCode* = 6. This will strip the text and form data from any No Cache pages, and so convert them back into short cache pages. This will respect the servers wish that you do not permanently cache those pages.

The Cache File structure:

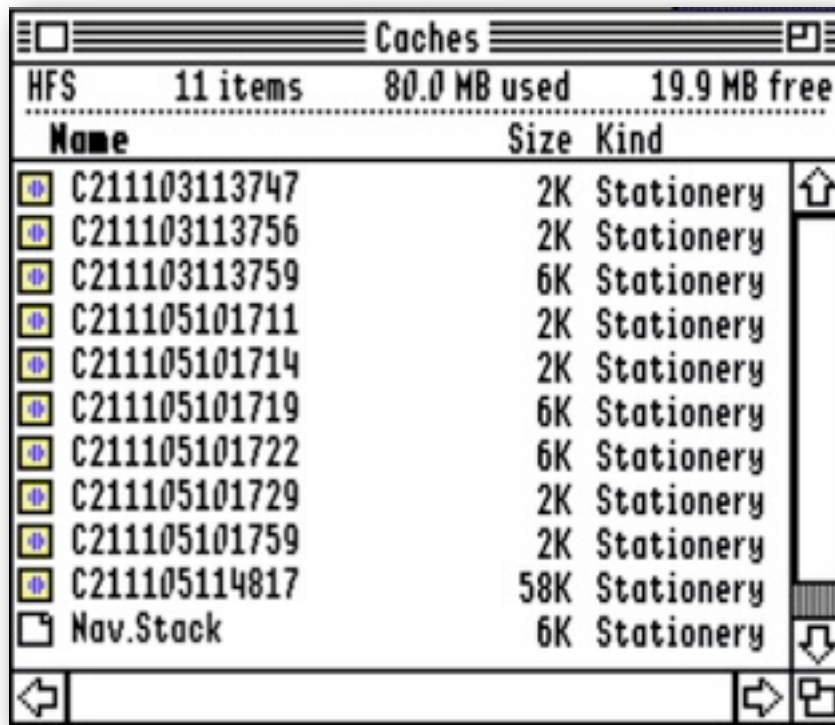
Cache File:

Offset to start of RefHandles (long)
cacheFlags (word)
 Bit 0 = If set a File://
 Bit 11 = Save pages as full caches (See No cache section)
 Bit 13 = Server required password (server request)
 Bit 14 = Server requested No Cache (Server request)
 Bit 15 = Short Cache (Application request)
userFlags (long) - for use by application
length of page retrieved from server (long)
page URL (pString 256 bytes)

Long cache section extends here:

TESelection (8 bytes)
length TEText (long)
TEText from control
length Style info (long)
Style Info from control
Form Active flag (word)
Number of RefHandles (long) then
 length of RefHandle (long) pair 1
 data from RefHandle pair 1
 length of RefHandle (long) pair 2
 data from RefHandle pair 2
 etc.

The Form *RefHandles* hold any entries you may have made to forms on a page, so when the page is later restored, that data will be restored as well as the page itself.



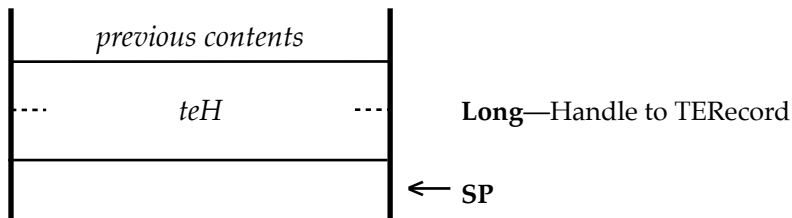
Caches			
HFS	11 items	80.0 MB used	19.9 MB free
Name	Size	Kind	
C211103113747	2K	Stationery	↑
C211103113756	2K	Stationery	
C211103113759	6K	Stationery	
C211105101711	2K	Stationery	
C211105101714	2K	Stationery	
C211105101719	6K	Stationery	
C211105101722	6K	Stationery	
C211105101729	2K	Stationery	
C211105101759	2K	Stationery	
C211105114817	58K	Stationery	
Nav.Stack	6K	Stationery	↓

\$1E82 HTMLSetTextTERecord

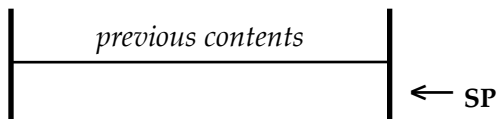
Replaces the text of a TERecord in the optional display window.

Parameters

Stack before call



Stack after call



Errors	\$82-2	htBadHandle	Invalid TERecord
	\$8204	htWindowClosed	Window closed
	\$8206	htNoData	The TERecord is empty
	\$8208	htNotInit	HTMLInit has not been called

C

```
extern pascal void HTMLInsertTERecord (teH);
Long      teH
```

This call is similar to HTMLInsertTERecord, but instead of inserting the text at the end of the record, it replaces the entire text with the contents of the passed TERecord. This saves first calling HTMLClearDisplay if you are replacing the entire contents of the display.

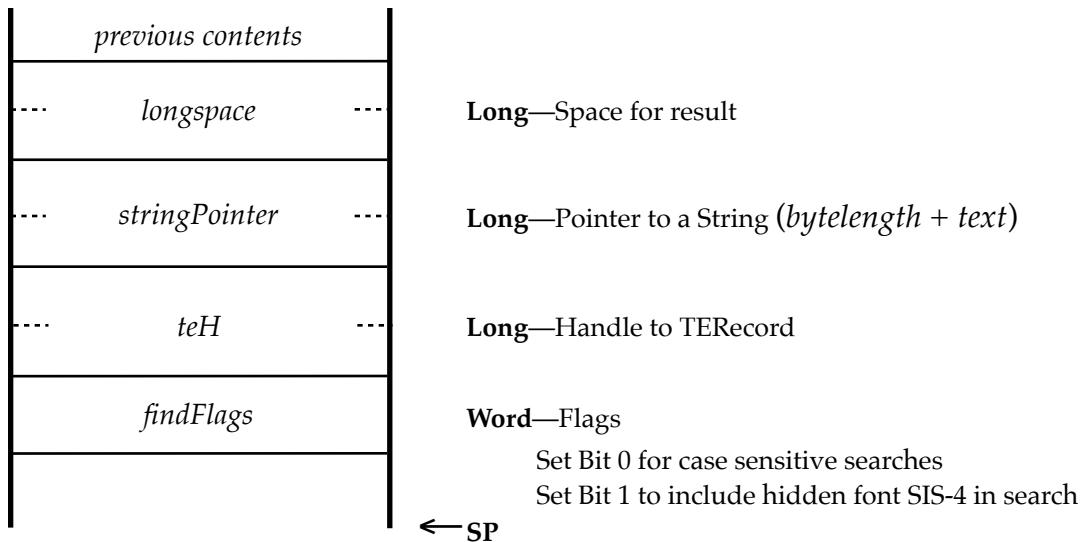
Note: The calling application must Kill the passed TERecord after the call.

\$1F82 HTMLFindString2

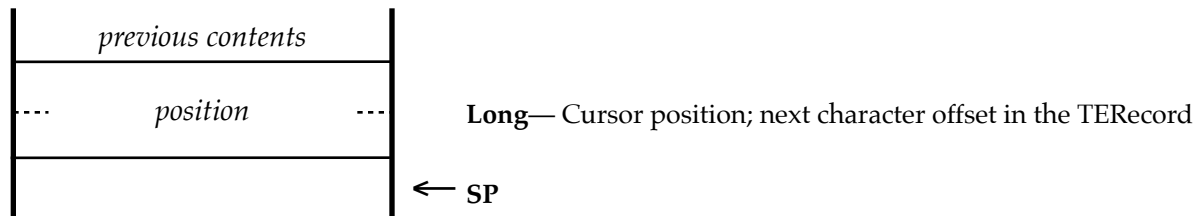
Finds the next occurrence of 'String', searching from the current cursor position.

Parameters

Stack before call



Stack after call



Errors	\$8204	htWindowClosed	Window closed
	\$8207	htFailed	'Name' String not found
	\$8208	htNotInit	HTMLInit has not been called
	\$82FF	htBadCall	Incorrect parameters supplied

C

```
extern pascal Long HTMLFindString (stringPointer);  
Long      position, stringPointer
```

This is similar to the HTMLFindString call, but allows you to specify the target TextEdit control, whether the search is case sensitive, and whether hidden text using the SIS-4 font is also searched.

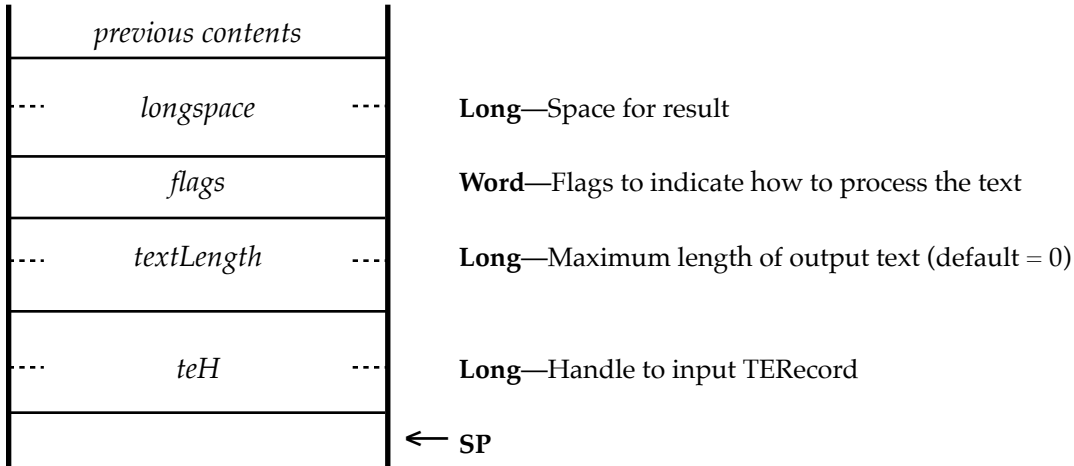
Note: If the target string is found, the text will be selected, and scrolled to place it in the middle of the screen.

\$2082 HTMLExtractText

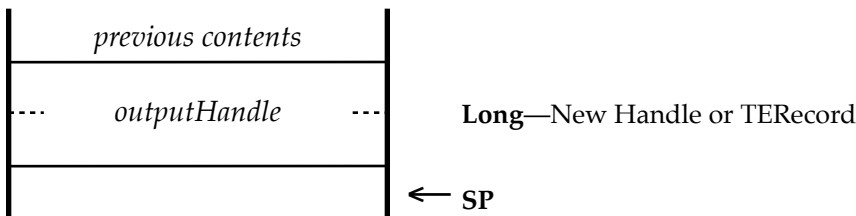
Extracts and prepares text from a TEREcord for Printing.

Parameters

Stack before call



Stack after call



Errors \$8202 htBadHandle A bad Handle was supplied

C extern pascal Long HTMLExtractText (flags, textLength, teH);
Long textLength, teH
Word flags

Web pages built using the HTML Tool, may contain any of the five special SIS fonts, and may also contain hard spaces. These fonts can cause problems when printing, either with characters that cannot be printed, or that unsuitable substitution fonts have to be used. By running the text through the *HTMLExtractText* call before printing, the problem fonts can either be ignored, or be changed to printable fonts, with hard spaces turned into normal spaces.

In addition, the *HTMLExtractText* call is able to process text to change coloured fonts to black and white, to force a single 10pt size, to strip font attributes, or to force the Shaston or Courier font.

The SIS font set:

SIS-1	proportional serif font
SIS-2	mono spaced serif font
SIS-3	icon font
SIS-4	hidden/invisible font
SIS-5	over/underlined button font
SIS-6	proportional sans-serif font
SIS-7	mono spaced sans-serif font

The *flags*:

Bit 0	Set to return text in a plain Handle
Bit 1	Set to strip coloured text to black on white
Bit 2	Set to strip font attributes to plain
Bit 3	Set to change all font sizes to 10 pt
Bit 4	Set to change all fonts to Helvetica
Bit 5	Set to change all fonts to Courier
Bit 6	Set to strip SIS-3 & SIS-4 fonts, and change others to printable fonts
Bit 7	Change any Hi-ASCII characters to \$20 printable spaces
Bit 9	Strip any text in the SIS-6 proportional sans-serif font
Bit 10	Strip any text in the SIS-7 mono-spaced sans-serif font
Bit 11	Strip any text in the SIS-1 proportional serif font
Bit 12	Strip any text in the SIS-2 mono-spaced serif font
Bit 13	Strip any text in the SIS-3 icons font
Bit 14	Strip any text in the SIS-4 hidden font
Bit 15	Strip any text in the SIS-5 button font

Using the call:

```

pha
pha
PushWord Flags
PushLong Length          max length of text to return 0 for all
PushLong TEControlHandle Handle to TE control record to extract from
_HTMLExtractText
PullLong ReturnedHandle  Handle or TEHandle of returned massaged text

```

To use the call, set *flags* appropriately, and pass the Handle of either a TextEdit control, or a TextEdit Record. On output, a TextEdit Record or Handle will be returned. This Handle can then be passed to your Print routine.

All the text from the input TextEdit control or TERecord will be processed. If you wish to only process selected text from a TE control, you must first extract that text and pass it in a new TERecord. Any hard spaces in the text \$CA, will be translated into normal printable spaces \$20.

Specify either the maximum length of output text you require, or pass a zero value to return all the usable input text to the output.

For a web page, the minimum setting for *flags* would be \$0040, which would strip the SIS-3 icon and SIS-4 hidden fonts, and change any other fonts into suitable printable fonts.

During the processing, Bits 9-15 of the *flags* (the seven strip flags) are processed first, followed by Bits 1-5, then Bit 6-7. This allows you to control precisely what will happen during the extraction.

The default output is to return a TE Record, which holds font information and styles, but optionally you can return just the plain text content in a Handle.

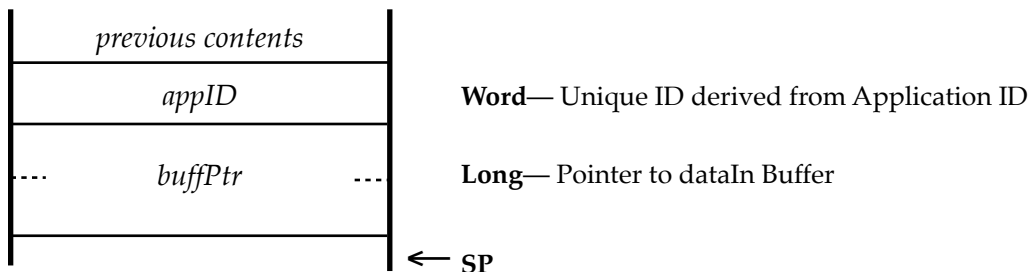
On return, the input TEHandle will be left untouched, but the calling application is responsible for disposing of the returned TE Record or plain Handle when it has finished with it.

\$2182 HTMLWriteToLog

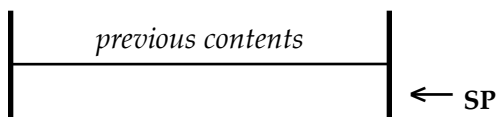
Allows a client to add comments or other text data to the Debug Log file.

Parameters

Stack before call



Stack after call



Errors None

C extern pascal void HTMLWriteToLog (buffPtr);
Long buffPtr, Word appID

The sole purpose of this call, is to allow web clients to add custom entries into the Debug Log file, such as details of Cookies that may have been deleted, or other actions the client may have taken.

dataIn buffer 26 bytes:

<i>textFormatFlags</i>	two bytes	<i>Flags controlling behaviour of the call</i>
<i>line1Length</i>	two bytes	<i>Length of Line 1</i>
<i>line1Pointer</i>	four bytes	<i>Pointer to text for Line 1</i>
<i>line2Length</i>	two bytes	<i>Length of Line 2</i>
<i>line2Pointer</i>	four bytes	<i>Pointer to text for Line 2</i>
<i>line3Length</i>	two bytes	<i>Length of Line 3</i>
<i>line3Pointer</i>	four bytes	<i>Pointer to text for Line 3</i>
<i>line4Length</i>	two bytes	<i>Length of Line 3</i>
<i>line4Pointer</i>	four bytes	<i>Pointer to text for Line 4</i>

If the length of any of the four lines is zero, then nothing will be written for that line. If the *textFormatFlags* is zero, and all the line lengths are zero, then nothing will be written to the file. Note that CRs are not printed automatically to the end of any of the lines, so if required, you will need to add those to the passed lines. Always add a double CR at the end of the text, either in your last line, or by setting Bit 14 of the *textFormatFlags*.

Defining the *textFormatFlags* bits:

Bit 0 = set for marker [S]
Bit 1 = set for marker [R]
Bit 2 = set for marker [F]
Bit 3 = set for marker [D]
Bit 4 = marker precedes text on line 1
Bit 5 = marker precedes text on line 2
Bit 6 = marker precedes text on line 3
Bit 7 = marker precedes text on line 4
Bit 13 = Single CR spacer at end of last line
Bit 14 = Double CR spacer at end of last line
Bit 15 = Divider line added at start of the entry

The Debug Log file uses an [S] marker to show whether data is being sent to a host server, or [R] if data is being received from a host server. You can precede your own lines with either an [S], [R], [F], or [D], by setting Bits 0-3 of *textFormatFlags*, then set Bits 4-7 to define which lines will have that marker applied. If the Bits 0-3 are zero, then a space [] marker will be printed.

Note that you must set either Bits 0, 1, 2, or 3, as the first marker to be set will always take precedence. If you wished to have different markers on each line of text, then you will need to make further calls to `HTMLWriteToLog` for each of the different markers.

If you have not included CRs at the end of the last line of text to be printed, then set Bits 13 or 14, to add either a single or a double CR at the end of all the lines of text.

Set Bit 15 if you wish to have the standard Debug Log divider line inserted at the very start of the text before any of the four possible text lines are printed.

This is an example of an `HTMLTextToLog` call, where an expired cookie has been deleted, and the client has shown the details using the `HTMLTextToLog` call:

```
[S] POST /test.php HTTP/1.1
User-Agent: Mozilla/2.0 (Compatible; Tool130; IIGs)
Host: test.com
Cookie: name=Steve
Content-Type: application/x-www-form-urlencoded
Content-Length: 25

[R] HTTP/1.1 200 OK
[R] Content-Type: text/html; charset=UTF-8
[R] Transfer-Encoding: chunked
[R] Connection: keep-alive
[R] Keep-Alive: timeout=15
[R] Date: Wed, 25 Aug 2021 10:47:30 GMT
[R] Server: Apache
[R] Set-Cookie: name=Steve; expires=Wed, 25-Aug-2021 10:52:31 GMT; Max-Age=300
[R]

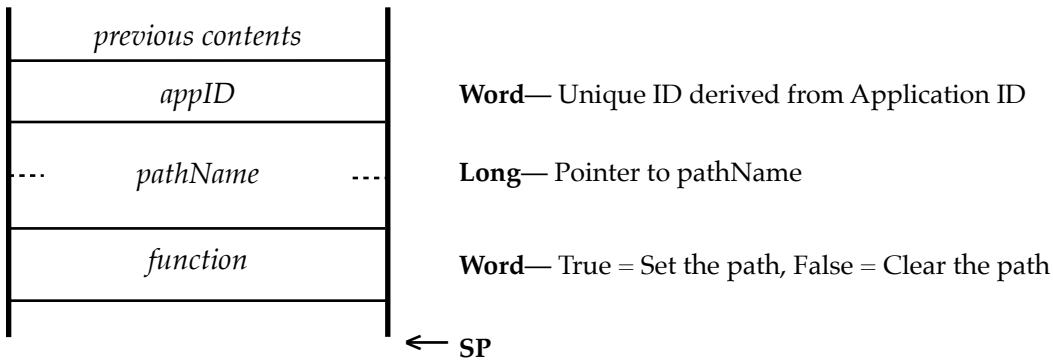
Cookie: K061261FBA01
Sent from: test.com was deleted
```

\$2282 HTMLSetLogPath

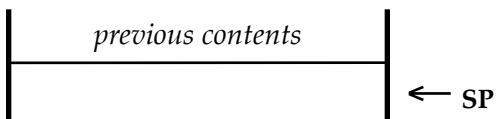
Sets the pathname that will be used for writing the Debug Log file.

Parameters

Stack before call



Stack after call



Errors

\$8207	htFailed	All 16 pathName slots are full
\$xxxx	GS/OS errors	The pathName was bad

C

```
extern pascal void HTMLSetLogPath (buffPtr);  
Long      pathName  
Word      appID, function
```

For the HTMLGetPage, HTMLPostForm, and HTMLWriteToLog calls, the default is to save the Debug data to an HTML.Log file in the top directory of the Boot drive.

The HTMLSetLogPath call allows a web browser to change that default pathname if required.

Only the pointers to the pathnames are stored in the Tool, so if a passed pathname is invalid, or becomes invalid at any point, the call will default to writing to the top level of the Boot drive as before.

Up to 16 separate pathname pointers can be stored, and these are linked by the *appID* passed in the call. This allows a browser to set up multiple pathnames for different purposes, by using different *appIDs* when calling HTMLSetLogPath, and then using those same *appIDs* for the HTMLGetPage, HTMLPostForm, and HTMLWriteToLog calls.

Set *function* to *true* when setting a path, and when your web browser Quits, or you are finished with that path, clear the entry by calling HTMLSetLogPath again with the same *appID*, and *function* set to *false*. Set the pathname pointer to *zero* when clearing the entry.

Cookies

The HTML Tool can optionally handle Cookies that are received, and then send them back to the Host as required. If the calling browser supports Cookies, set Bit 3 of the *inputSettings* or *formPrefs* flag, and make sure that the appropriate *dataIn* and *dataOut* buffers have the extra four bytes at the end to hold the Handles to be passed back and forth.

For the *HTMLGetPage* and *HTMLPostForm* calls, if Bit 3 of the *inputSettings* or *formPrefs* flag is set, the HTML Tool will return any Cookies it receives from the server in the Incoming Handle. If the Incoming Handle value is zero, then no Cookies were returned.

For the *HTMLGetPage* and *HTMLPostForm* calls, if Bit 3 of the *inputSettings* or *formPrefs* flag is set, and a valid Handle is supplied in *cookieList*, the contents of the Handle will be sent along with the Header data to the server. If a Handle value is zero, no Cookie will be sent.

Note: Cookies are not sent if the HEAD only flag has been set for the *HTMLGetPage* call.

Apart from the Host URL that will be added to the start of the Incoming Handle, no processing of the Cookie data is made by the HTML Tool before returning any Cookies that have been received. It will be up to the browser to process the Incoming Cookies appropriately, and to then send them back correctly formed as and when required.

The browser must delete the Incoming Handle when it has finished with it, keeping the Cookie data as required. The HTMLTool deletes the Outgoing Handle after it has been sent.

Incoming Handle

<i>hostURL</i>	<i>pString</i>	Host URL pString terminated with a CR
<i>cookie1</i>	data	each Cookie is terminated by a CR
<i>cookie2</i>	data	etc.

The browser must remove and appropriately evaluate any attributes in the Cookie before constructing the outgoing Handle to be sent back.

For example, if five Cookies were received from the host, the returned Handle contents might look like this. Note that each Cookie is separated by a CR, and there is also a trailing CR:

```
<host-name pString>CR<cookie1-name>=<cookie-value>CR<cookie2-name>=<cookie-value>;  
Expires=<date>CR<cookie3-name>=<cookie-value>; Max-Age=<number>CR<cookie4-  
name>=<cookie-value>; Domain=<domain-value>CR<cookie5-name>=<cookie-value>;  
Path=<path-value>CR
```

Outgoing Handle

<i>cookie1</i>	data	Cookie is terminated by ' ; ' semicolon and space
<i>cookie2</i>	data	etc.

If the five Cookies above were being sent back, this example shows the data in the outgoing Handle. Note the *hostURL* has been removed, and each Cookie has a trailing ' ; ' (semi-colon and space) separating it from the previous Cookie. There is no trailing ' ; ' at the end:

```
<cookie1-name>=<cookie-value>; <cookie2-name>=<cookie-value>; <cookie3-  
name>=<cookie-value>; <cookie4-name>=<cookie-value>; <cookie5-name>=<cookie-  
value>
```

Example of posting a form

HTML Code:

```
<!-- Form which will send a POST request to the current URL -->
<form method="post">
  <label>Name:
    <input name="submitted-name" autocomplete="name">
  </label>
  <button>Save</button>
</form>
```

Screen Display:



HTTP POST transcript data sent:

```
POST /speccie/downloads/sample.html HTTP/1.1
User-Agent: Mozilla/2.0 (Compatible; Versions 1.0; IIgs)
Host: speccie.uk
Content-Type: application/x-www-form-urlencoded
Content-Length: 47

submitted-name=This+is+a+test+of+Posting+a+form
```

Data returned:

```
HTTP/1.1 200 OK
Date: Tue, 19 May 2020 17:47:16 GMT

<html><body>
The requested page will be returned here. The server may have changed the data.
</body></html>
```

HTTP GET transcript of the same data sent:

```
GET /speccie/downloads/sample.html? submitted-
name=This+is+a+test+of+Posting+a+form HTTP/1.1
User-Agent: Mozilla/2.0 (Compatible; Versions 1.0; IIgs)
Host: speccie.uk
Pragma: no-cache
Accept: /*/*
```

Data returned:

```
HTTP/1.1 200 OK
Date: Tue, 19 May 2020 17:47:16 GMT

<html><body>
The requested page will be returned here. The server may have changed the data.
</body></html>
```

Rolling your own Web Browser

With the addition of the new HTMLGetPage call, and the existing HTML Tool calls, you can now painlessly build your own web browser. Refer to each call for further details.

First build a window template that has a TextEdit Control with the ID \$7000, and pass a pointer to this template to HTMLOpenDisplay2 (Page 28). This will open the interactive window that will display the incoming pages. As well as any other controls you have in the template, optionally you can include a Stop button or Stop icon button with the ID \$8000.

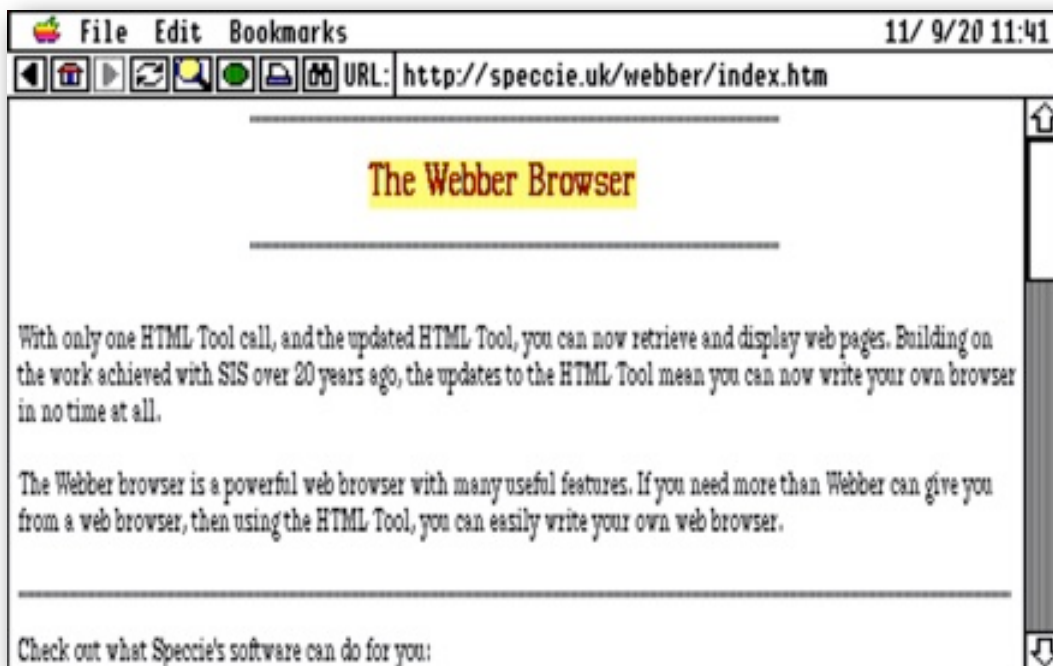
Now pass a valid target URL to HTMLGetPage (Page 30), and pass the returned TextEditHandle to HTMLInsertRecord (Page 16).

That's all there is to it. You have now retrieved and displayed a parsed <html> page into your very own Web Browser!

In a doModalWindow loop, whenever you detect a double-click, call HTMLEventToDisplay (Page 29), and process the returned event according to the values you see (Page 18). It could be a link to another page, or a Form button that had been double-clicked. If it was another page, just call HTMLGetPage and HTMLInsertRecord again, but if it was a Form button, then call HTMLPostForm with the Form number (Page 32).

If you are keeping a navigation history stack, use the HTMLCacheControl call to Save, Update, Restore or Kill, pages from your Cache folder.

Of course you will need to add considerable supporting code in addition to those calls, but that is true of any application you may write. To give you an example of what can be done with these few calls, please checkout the Webber browser. This is a powerful web browser that was built to test out the new calls, and was built exactly as I have described above!



Using the HTML Color tags

The HTML Tool supports two Font tags that can control the color of the text and its background.

Use the tags in this way, remembering to balance each tag with its corresponding end tag:

```
<font color="red">This text will be in Red</font>  
<font bgcolor="yellow">Black text with a Yellow background</font>  
<font color="blue"><font bgcolor="lightgreen">Blue text with a Lightgreen background  
</font></font>
```

The 16 IIGs colors are mapped to these HTML tag colors:

black	= black
blue	= blue, navy
darkgreen	= darkgreen, olive
darkgray	= darkgray, gray
red	= red, maroon
lilac	= lilac, purple
orange	= orange
pink	= pink, fuchsia
green	= green, lime
aqua	= aqua, cyan
lightgreen	= lightgreen, chartreuse
lightblue	= lightblue, teal
lightgray	= lightgray, silver
periwinkle	= cornflowerblue
yellow	= yellow
white	= white

Note: Due to the limitations of the SHR display, not all the colors will be easily readable unless the size is increased using one of the Headline tags, or using the Bold tag.

Experiment with the colors using Spectrum™, by adding an .HTM suffix to the file, and opening the file in the Spectrum™ Editor by choosing to display as HTML. Alternatively, if the text file holds an <html> tag, you will be given the choice to open as HTML, without the suffix.

This is a black line
This is a blue line
This is a navy line
This is a darkgreen line
This is an olive line
This is a darkgray line
This is a gray line
This is a red line
This is a maroon line
This is a lilac line
This is a purple line
This is an orange line
This is a pink line
This is a fuchsia line
This is a green line
This is a lime line
This is an aqua line
This is a cyan line
This is a lightgreen line
This is a chartreuse line
This is a lightblue line
This is a teal line
This is a lightgray line
This is a silver line
This is a cornflowerblue line
This is a yellow line

This text will be in Red

Black text with a Yellow background

Blue text with a Lightgreen background

Using the HTML Custom tags

The HTML Tool supports custom tags to support the SIS font set, playing System sounds, and speaking text with the ByteWorks Talking Tools.

You can play a system sound when the icon button is clicked by including the name of the system sound in a line such as this:

```
<object sis-sound="Trumpets"> Lets's hear it for Webber... </object>
```

To play talking text, you must have the speech Talking Tools installed. You can obtain the Talking Tools, along with other ByteWorks software, on the OPUS II disk from the Juiced.GS Store:

<https://juiced.gs/store/opus-ii-software/>

Include the text within the object tag, and add an optional description before you close the tag:

```
<object sis-speech="They said this Brauwerr could not be done.">Welcome to Webber</object>
```

You can control the voice timbre with additional commands:

```
<object sis-speech="Text" gender="female">Description</object>
```

```
<object sis-speech="Text" gender="male">Description</object>
```

```
<object sis-speech="Text" pitch="1" speed="1" tone="1">Description</object>
```

You may need to experiment with these settings for the best results, and remember that an accelerated or emulated IIGs may sound very different to a standard speed IIGs.

To display text directly using the seven custom SIS fonts, use tags <f1> to <f7>. Apart from the size attribute, these tags work in the same way as the tag, so close them with :

```
<f1>Set the SIS proportional serif font</font>
```

```
<f2>Set the SIS mono-spaced serif font</font>
```

```
<f3>Set the SIS symbol font</font>
```

```
<f4>Set the SIS invisible font</font>
```

```
<f5>Set the SIS over/underlined button font</font>
```

```
<f6>Set the SIS proportional sans-serif font</font>
```

```
<f7>Set the SIS mono-spaced sans-serif font</font>
```

It is suggested you use a font editor to see what the SIS fonts will actually display, and for the symbol font, which ASCII value is needed for the symbol you want to use.

If you can't use the required character or symbol directly within the text, use the Ampersand function, with a value of 256 added to the ASCII value, to display the required character:

EG: Ł will print a capital A, as the ASCII value of A is 65, so with 256 added, we get 321.



HTML Tool Set error codes

\$8201	htNoFonts	Required fonts not installed
\$8202	htBadHandle	Bad Handle value supplied
\$8203	htWindowOpen	Display window already open
\$8204	htWindowClosed	Display window already closed
\$8205	htBadParms	Bad parameters were passed
\$8206	htNoData	Data was not present when expected
\$8207	htFailed	The call failed to execute
\$8208	htNotInit	HTMLInit has not been called
\$8209	htNoTCPIP	TCP/IP not installed
\$820A	htBadPath	A bad URL was provided
\$820B	htGenError	General error with online calls
\$820C	htUserKilled	User stopped process
\$82FE	htNoTools	Required Tools not installed
\$82FF	htBadCall	The call failed through bad Data

Errors returned if main error is htGenError:

\$0001 = Bad Connection

\$0002 = No response

\$0004 = Bad response

\$0008 = GS/OS error file error or would not open

\$0010 = GS/OS write error

\$0020 = GS/OS volume full

\$0040 = TCP/IP DNR abort error

\$8000 = User killed/stop

Links

To obtain the HTML Tool Set, and any of my other software:

<http://speccie.uk>

To subscribe to the Juiced.GS magazine:

<http://juiced.gs/>

To read all about the annual KFest conference:

<http://www.kansasfest.org/>