RTIP

WEBC MANUAL

TABLE OF CONTENTS

Revised 8/3/05

CHAPTER 1

INTRODUCTION TO WEBC

1.1 INTRODUCTION TO WEBC

Welcome to WebC, EBSnet's HTML based graphical user interface SDK. WebC is based on three primary data structures: the HTML Browser, the HTML Document, and the HTML Element. Much of the WebC API consists of functions that perform operations on these three objects. This API also contains a host of additional functions to support features such as C function event handlers, cookie management, and JavaScript extensions.

The first step to utilizing the full potential of WebC is to understand what the three basic types represent, and how they fit in to the larger picture.

1.1.1 HTML BROWSERS

HTML Browsers are the most general, root-level object in WebC. They correspond to individual HTML viewing windows. HTML Browsers have functions for URL navigation and controlling various window properties, as well as the ability to attach user data for application specific use. HTML Browsers are referred to in the WebC API by their HBROWSER_HANDLE.

1.1.2 HTML DOCUMENTS

HTML Documents are the internal representation of an HTML source document. There is always at least one HTML Document per HTML Browser, which is called the Root Document. HTML Documents are referred to in the WebC API by their HDOC_HANDLE.

1.1.3 HTML ELEMENTS

HTML Elements are nodes of a tree structure that is contained within the HTML Document. This structure is called the Document Tree. The structure of the Document Tree mirrors the nesting structure of the tags within an HTML source document. Tags and text fragments in the HTML source document are both represented by HTML Elements. The HTML Document always contains at least one HTML Element, the Body Element, which is the root of the Document Tree. HTML Elements are referred to in the WebC API by their HELEMENT_HANDLE.

CHAPTER 2

BUILDING/CONFIGURING WEBC

2.1 BUILDING WEBC

WebC is designed to be built using command line tools and compilers. This makes it relatively easy to port to other architectures and platforms, as the same makefile can be used with multiple compilers with little modification.

WEBC RELEASE TREE DIRECTORIES AND DESCRIPTIONS OF THEIR CONTENTS

SOURCE\
This directory contains all of WebC's source files.

INCLUDE\
All WebC header files referenced from the source are stored here.

MOZILLA\
WebC uses the Mozilla project's SpiderMonkey JavaScript implementation. All related source files are stored in this directory.

BIN\
Programs used during compilation are stored here. These include gmake, the GNU Make utility; docxx, a source file documentation program; and bintoc, a program used to compile arbitrary files into C source code.

FONTS\
All of the fonts WebC uses are created with the PEG Font Capture utility, and stored in this directory.

HTML\
All HTML files in this directory will be compiled with bintoc, and then linked into the WebC library file. These files can be accessed within the program via the WEBC protocol (ie webc://file.html).

MAKE\
Platform independent make settings are stored in this directory. Currently, the only file stored here is mkwebc.inc. This file contains definitions for all of WebC's modules.

Additionally, each target architecture and compiler has its own directory. In other words, binaries compiled for Win32 with Visual C++ will be in a different directory than binaries compiled for Win32 with the Borland C++ compiler. All compiler and architecture specific configuration files are kept in their respective directories.

FILES COMMON TO ALL THE TARGET DIRECTORIES

MAK.BAT
This batch file starts the build process. Both WebC and PEG libraries will be compiled.

CLEAN.BAT
Delete all binary and intermediate files.

SETVARS.BAT
Set all environment variables. This batch file must be run before compilation.

RULES.INC
Compiler specific compilation configuration. Includes rules for compiling each type of file, and compiler specific dependencies and libraries.

MAKEFILE
Contains rules for each make target: doc, clean, peg, webc and all. doc is used to build source code documentation with docxx, and clean is equivalent to calling clean.bat. The peg and webc targets build the respective libraries, while all builds both.

SUB DIRECTORIES CONTAINED IN IN EACH TARGET DIRECTORY

BIN\
All executables are compiled to this directory

LIB\
The WebC and PEG library files are stored here upon compilation.

TEMP\
.c and .h files created with bintoc are stored here.

OBJ\
Intermediate compiler output files are compiled to this directory.

Building the WebC library involves first calling setvars.bat, and then mak.bat from within the target directory.

Note: Win9x machines might need the command interpreter started with the /e option to ensure that there is enough environment space to store all of the variables instantiated by setvars.

2.2 CONFIGURING WEBC

2.2.1 WEBC CONFIGURATION OPTIONS

WebC is configured entirely at compile time. This prevents unnecessary modules from being compiled, thus reducing the memory footprint of your application.

JavaScript and HTTPS (SSL) are enabled and disabled through setvars.bat. Set the respective envirnonment variable to non-zero to enable, or zero to disable. Nearly all other configuration options can be found in webcfg.h, as pre-processor definitions. An option is enabled if it is defined to be non-zero.

WEBC_SUPPORT_SOCKETS
Socket support – required for all network protocols.

WEBC_SUPPORT_USER_EVENTS
Support custom user event handlers.

WEBC_SUPPORT_STYLE_SHEETS
Enable this option to compile in support for level one CSS.

WEBC_CFG_MAX_EVENT_HANDLERS
Maximum number of custom user event handlers. Has no effect unless user events are enabled.

WEBC_SUPPORT_COOKIES
Enable or disable the use of cookies.

WEBC_SUPPORT_PIXEL_SCALING
Support re-sizing of output for small displays.

WEBC_CFG_PIXEL_MUL
When scaling is enabled, this is the numerator of the pixel scaling factor of the display device.

WEBC_CFG_PIXEL_DIV
When scaling is enabled, this is the denominator of the pixel scaling factor of the display device.

WEBC_SUPPORT_URL_MACROS
Enable / disable URL macros.

WEBC_SUPPORT_BLANK_TARGET
Enable to allow anchors to point to _blank.

WEBC_SUPPORT_STORE_BITMAP
Enable to allow the storing and retrieving of bitmaps. Required for image caching.

WEBC_SUPPORT_HTTP
Support the HTTP protocol (HTTP://...).

WEBC_SUPPORT_FILE
Support the FILE protocol (FILE://...).

WEBC_SUPPORT_FTP
Support the FTP protocol (FTP://...). Not currently implemented

WEBC_SUPPORT_MAILTO
Allow URLs to invoke a mail program (MAILTO:...). Not currently implemented

WEBC_SUPPORT_INTERNAL
Enable to support the precompiled file system (WEBC://...).

WEBC_SUPPORT_IMAP
Not currently implemented

WEBC_SUPPORT_POP3
Not currently implemented

WEBC_SUPPORT_WEBS
Not currently implemented

WEBC_SUPPORT_TABLES
Enable to support the <TABLE> tag.

WEBC_SUPPORT_FRAMES
Enable to support multiple frames in one document.

WEBC_SUPPORT_IMAGES
Enable to support the <IMG> tag.

WEBC_SUPPORT_GIFS
Enable to support the reading and displaying of .GIF files.

WEBC_SUPPORT_JPGS
Enable to support the reading and displaying of .JPG files.

WEBC_SUPPORT_BACKGROUND_IMAGES
Enable to support background images.

WEBC_CFG_HISTORY_SIZE
Size of the browser history.

WEBC_CFG_EVENT_QUEUE_SIZE
Maximum number of events that may be enqueued.

WEBC_CFG_EVENT_MAX_RECURSE_DEPTH
Maximum depth the event handler may recurse to.

WEB_CFG_MAX_COOKIE_BYTES
Maximum size of any cookie.

WEBC_CFG_META_TABLE_SIZE
need description

WEBC_CFG_MAX_FONT_FAMILIES
Maximum number of font families.

WEBC_CFG_MAX_STREAMS
Maximum number of open network streams.

WEBC_DRAW_IMAGE_BOXES
Enable to draw empty boxes for images that have not been loaded yet.

WEBC_MINIMIZE_SCREEN_REFRESH
need description

WEBC_USE_3D_FRAMES
Draw right / bottom and top / left borders different shades of grey, giving them a 3D look.

WEBC_ANIMATE_GIFS
Enable to render animated .GIFs correctly, disable to only render their first frame.

WEBC_INCLUDE_NET_CFG_DIALOG
need description

WEBC_HOMEPAGE
A URL string that will be opened upon WebC startup.

WEBC_SUPPORT_VERIFY
Enable to verify certificates received over HTTPS.

WEBC_SUPPORT_FONTS
Enable to support the <FONT> tag.

WEBC_DEFAULT_FONT
This macro should evaluate to a pointer to the default font.

WEBC_SUPPORT_ANCHORS
Enable to support the <A> tag.

WEBC_SUPPORT_CACHE
Enable to include support for text and image caching.

WEBC_CACHE_IMAGES
Enable to support image caching. Requires that WEBC_SUPPORT_CACHE be enabled.

WEBC_CFG_CACHE_BLOCKS
need description

CACHE_IMG_EXPIRE_TIME
Number of seconds to keep images in the cache.

WEBC_CACHE_BLOCK_SIZE
The size of each cache block, in bytes.

WEBC_SUPPORT_BODY_EVENTS
Allow the body of an HTML document to receive events. Only has an effect if JavaScript is enabled.

WEBC_MEMORY_DEBUG
Enable to log all memory operations to a socket.

WEBC_BUFFER_GUARD
Number of bytes to buffer each memory allocation with.

WEBC_GUARD_BYTE
The guard buffer will be cleared to this byte if memory debug is enabled.

WEBC_LOAD_BUFFER_SIZE
Amount of memory to allocate for the HTML parser.

WEBC_IMAGE_BUF_SIZE
need description

WEBC_TMO
Timeout for network connections, in seconds.

WEBC_MAX_USERNAME_LEN
Maximum length for the anchor component of a URL.

WEBC_MAX_HOSTNAME_LEN
Maximum length for the anchor component of a URL.

WEBC_MAX_PATHNAME_LEN
Maximum length for the path component of a URL.

WEBC_MAX_ANCHOR_LEN
Maximum length for the anchor component of a URL.

WEBC_DEFAULT_INPUT_SIZE
Default size of the text box for the <INPUT> tag.

WEBC_TITLE
String to render to the title bar.

WEBC_DEFAULT_LINK_COLOR
Default link color, of the form {Red, Green, Blue}. Each color value is a number in the range 0-255.

WEBC_DEFAULT_VISITED_LINK_COLOR
Default visited link color, of the form {Red, Green, Blue}. Each color value is a number in the range 0-255.

WEBC_DEFAULT_ACTIVE_LINK_COLOR
Default active link color, of the form {Red, Green, Blue}. Each color value is a number in the range 0-255.

WEBC_DEFAULT_BACKCOLOR
Default background color, of the form {Red, Green, Blue}. Each color value is a number in the range 0-255.

WEBC_HR_HEIGHT
The default height of a horizontal rule created with the <HR> tag.

WEBC_LINE_SPACING
Number of pixels to vertically pad lines of text.

WEBC_LEFT_BORDER
Border thickness, in pixels, of the left of the HTML window.

WEBC_RIGHT_BORDER
Border thickness, in pixels, of the right of the HTML window.

WEBC_TOP_BORDER
Border thickness, in pixels, of the top of the HTML window.

WEBC_BROKEN_IMG_LINK_WIDTH
The width of the box that is rendered for images that can not be loaded.

WEBC_BROKEN_IMG_LINK_HEIGHT
The height of the box that is rendered for images that can not be loaded.

WEBC_DEFAULT_TEXT_INPUT_LEN
need description

WEBC_DEFAULT_TEXTAREA_COLS
Default number of columns to create for the <TEXTAREA> tag.

WEBC_DEFAULT_TEXTAREA_ROWS
Default number of rows to create for the <TEXTAREA> tag.

2.2.2 WEBC FONT FAMILIES

All fonts used by WebC are pre-rendered. A font family consists of all seven size renderings of a font, in normal, bold, italic and boldfaced italic. The declarations of each of these renderings are made in webfonts.cpp. Next, each of these declarations are placed into a global array. Finally, instantiate a global WebcFontFamily object, initialized with the font type name, generic font family name, and a pointer to the font table. For example, the Sans-Serif font is defined as follows:

```
extern  PegFont
    Sans1,Sans1b,Sans1i,Sans1bi,
    Sans2,Sans2b,Sans2i,Sans2bi,
    Sans3,Sans3b,Sans3i,Sans3bi,
    Sans4,Sans4b,Sans4i,Sans4bi,
    Sans5,Sans5b,Sans5i,Sans5bi,
    Sans6,Sans6b,Sans6i,Sans6bi,
    Sans7,Sans7b,Sans7i,Sans7bi;

PegFont  *gpSansFontTable [] = {
    &Sans1,&Sans1b,&Sans1i,&Sans1bi,
    &Sans2,&Sans2b,&Sans2i,&Sans2bi,
    &Sans3,&Sans3b,&Sans3i,&Sans3bi,
    &Sans4,&Sans4b,&Sans4i,&Sans4bi,
    &Sans5,&Sans5b,&Sans5i,&Sans5bi,
    &Sans6,&Sans6b,&Sans6i,&Sans6bi,
    &Sans7,&Sans7b,&Sans7i,&Sans7bi
};
WebcFontFamily  gArialFamily(“Arial”,
“sans-serif”,  gpSansFontTable);
```

The font data file will usually be created with the PEG Font Capture Utility, then compiled and linked with the program.

CHAPTER 3

WEBC PORTING GUIDE

3.1 INTRODUCTION

Porting the WebC development kit to alternate platforms simply requires implementing three operating system specific modules — The network socket layer, the file system interface, and several timing functions.

3.2 THE SOCKET LAYER

All of WebC's low-level network functionality is enapsulated in the socket layer. Since the WebC socket interface is very similar to WinSock and Berkely Sockets, implementing the socket layer is usually as simple as writing a few wrapper functions.

INT WEBC_NETWORK_INIT
(void)

Initialize the network interface. Perform any one-time initialization here.

VOID WEBC_NETWORK_CLOSE
(void)

Shut down the network interface.

INT WEBC_SOCKET
*(SOCKET * pSockDesc)*

Create a non-blocking socket. The newly created socket should be returned in the pSockDesc variable.

VOID WEBC_CLOSESOCKET
(SOCKET SockDesc)

Close the specified socket.

INT WEBC_CONNECT
*(SOCKET SockDesc, char * ip, RTP_UINT16 port)*

Attempt to establish a TCP connection to the specified IP address and port.

INT WEBC_RECV
*(SOCKET SockDesc, char * buffer, long size, webcIdleFn IdleFunc, char * IdleData)*

Receive data from an open socket. The data will be placed into buffer, up to a maximum of size bytes. The IdleFunc parameter is a pointer to a callback function that should be called between poll cycles. It's single argument should be IdleData. The call to webc_recv must block the thread it is called in until either the buffer is filled, the operation times out, or IdleFunc returns failure.

INT WEBC_SEND
*(SOCKET SockDesc, char * buffer, long size, webcIdleFn IdleFunc, char * IdleData)*

Send data over an open socket. webc_send will send size bytes from buffer through the specified socket. As in webc_recv, IdleFunc should be called with IdleData as its argument between poll cycles, and the function should return failure if the data can not be sent or upon the failure of the call to IdleFunc.

INT WEBC_GETHOSTIPADDR
*(char * ipAddress, char * host)*

Convert the hostname host to a four byte IP address, to be returned in ipAddress. host can be either a dotted IP string ("192.168.0.1") or a domain name ("www.stuff.com").

All functions that return a value should return zero on success, and less than zero on an error.

3.3 THE FILE SYSTEM INTERFACE

3.3.1 FILE SYSTEM INTERFACE FUNCTIONS

WebC needs access to the operating system's file system to maintain a local cache, keep track of SSL certificates, and to support the FILE protocol. The interface consists of the following functions, which follow closely to C's file I/O library:

INT WEBC_FOPEN
*(char * fileName, WEBC_FILE * FileHandle, int mode)*

Open the file specified by fileName. The resulting handle will be returned in FileHandle. Mode will be one of the following:

WEBC_FILE_RDONLY	- Open the file as read-only
WEBC_FILE_WRONLY	- Open the file for writing
WEBC_FILE_APPEND	- Open the file for writing, but do not truncate the file if it already exists
WEBC_FILE_UPDATE	- Open the file for reading and writing

INT WEBC_FCLOSE
(WEBC_FILE FileHandle)

Close a file previously opened with webc_fopen.

INT WEBC_FREAD
*(WEBC_FILE FileHandle, char * buffer, long size)*

Read from a file opened with webc_fopen. size bytes of data will be read into buffer.

INT WEBC_FWRITE
*(WEBC_FILE FileHandle, char * buffer, long size)*

Write to a file opened with webc_fopen. size bytes of data from buffer will be written at the current position in the file.

INT WEBC_FSEEK
(WEBC_FILE FileHandle, long offset, int from)

Move the read/write position offset bytes from the current position.

Note: As with the socket interface, all functions should return zero on success and negative on failure.

3.4 TIMING

3.4.1 TIMING FUNCTIONS USED BY THE USER INTERFACE TO DETECT EVENTS

The timing functions are used by the user interface to detect events such as double clicks and dragging, and by the network layer to generate random numbers. The following functions need to be defined and implemented:

EBS_GET_SYSTEM_TIME
*(EBSTIME * pTime)*

This function should fill in the year member of pTime with the current year, and the second field with the total number of seconds elapsed since January first.

LONG KS_GET_TICKS
(void)

Return the number of ticks since system startup.

LONG KS_MSEC_P_TICK
(void)

Return the number of milliseconds per tick.

LONG KS_TICKS_P_MSEC
(void)

Return the number of ticks per millisecond.

VOID KS_SLEEP
(long ticks)

Sleep for the specified number of clock ticks.

CHAPTER 4

WEBC USER GUIDE

4.1 INTRODUCTION TO WEBC

Welcome to WebC, EBSnet's HTML-based graphical user interface SDK. WebC is based on three primary data structures: the HTML Browser, the HTML Document, and the HTML Element. Much of the WebC API consists of functions that perform operations on these three objects. This API also contains a host of additional functions to support features such as C function event handlers, cookie management, and JavaScript extensions.

The first step to utilizing the full potential of WebC is to understand what the three basic types represent, and how they fit in to the larger picture.

4.1.1 HTML BROWSERS

HTML Browsers are the most general, root-level object in WebC. They correspond to individual HTML viewing windows. HTML Browsers have functions for URL navigation and controlling various window properties, as well as the ability to attach user data for application-specific use. HTML Browsers are referred to in the WebC API by their HBROWSER_HANDLE.

4.1.2 HTML DOCUMENTS

HTML Documents are the internal representation of an HTML source document. There is always at least one HTML Document per HTML Browser, which is called the Root Document. HTML Documents are referred to in the WebC API by their HDOC_HANDLE.

4.1.3 HTML ELEMENTS

HTML Elements are nodes of a tree structure that is contained within the HTML Document. This structure is called the Document Tree. The structure of the Document Tree mirrors the nesting structure of the tags within an HTML source document. Tags and text fragments in the HTML source document are both represented by HTML Elements. The HTML Document always contains at least one HTML Element, the Body Element, which is the root of the Document Tree. HTML Elements are referred to in the WebC API by their HELEMENT_HANDLE.

4.1.4 CAPABILITIES OF WEBC

WebC allows the application programmer to assign C functions to individual HTML Elements, to be called to handle events that happen to those elements. For further discussion of C function event handling, see **Working with Custom Event Handlers**.

WebC lets the application programmer assign C callback functions for the handling of unrecognized META tags. See **Providing Custom Handling for META Tags** for more information.

WebC provides a way to define macros to be expanded in URLs found in the source HTML (the href attribute of an anchor, for example) before those URLs are used to retrieve resources (either locally or across a network). See the section **Defining URL Macros** for more information on this topic.

WebC also defines a custom protocol, WEBC://, which is used to access pre-compiled data files located in a virtual file table. The application programmer can statically add entries to this table at compile-time, or dynamically create "virtual files" at run-time through the WebC API (see **Using WebC Virtual Files**).

In addition, WebC provides a set of API calls to manage HTTP cookies for communicating client-state information to specific hosts. See **Using WebC to Manage Cookies** for a discussion of cookies and how to use them within WebC.

Finally, WebC allows the application programmer to define native C functions that are accessible through the JavaScript interpreter. This provides a powerful, extensible programming capability to the WebC SDK. See **Defining New JavaScript Functions from C** for a discussion of how to create JavaScript native functions.

4.2 INITIALIZING THE WEBC LIBRARY

Before any WebC API calls can be made, webc_Init must be called to initialize the WebC library. After this call, all the functions of WebC will be available to the user. Many functions that require an HTML Browser, Document, or Element handle as a parameter, however, will not work until the application programmer has created one or more HTML Browsers by calling webc_CreateBrowser.

When a program is done using WebC, it should call webc_Exit. The resources initialized in webc_Init will not be freed until webc_Exit is called a number of times equal to the total number of calls made to webc_Init. It is, therefore, acceptable good practice to nest calls to webc_Init in various places in an application who might not be aware that the others have called webc_Init. Because nesting is accounted for, the WebC library will remain initialized and usable until the last part of an application has released claim on the library through webc_Exit.

4.3 WORKING WITH HTML BROWSERS

4.3.1 INTRODUCTION

The HTML Browser, as noted above, is the root level data object of the WebC library. Each HTML Browser instance corresponds to a separate HTML display window, optionally with a graphical button toolbar, title bar, URL entry area, and status bar.

HTML Browsers can execute concurrently (multiple browser windows displayed and operational at once, although see the qualification regarding load jobs in the next section) or modally (a single browser blocks the thread that created it until it is closed via webc_DestroyBrowser in response to some event, possibly user input). Modal execution is useful for implementing dialog-box style windows in WebC.

Each HTML Browser has its own load queue for downloading URL resources, such as HTML documents, images, style sheets, and JavaScript files. Only one Browser's load queue can be executing load jobs in a single thread at once (currently, WebC only supports single-threaded operation). Therefore, if a URL load in a particular Browser window is initiated while another Browser's load is in progress, WebC will check how many Browser's are in a load state, and if it is under a preset limit, then the new Browser load will proceed and complete before the currently loading Browser continues its load. If this limit is exceeded, then the Browser will be placed at the end of a queue of Browsers which have pending load jobs, and its jobs will be loaded when all the Browsers in front of it have completed their load. The limit on simultaneously loading HTML Browsers is the **WEBC_CFG_MAX_LOAD_NESTING** macro.

A URL load may be terminated before it is completed by clicking the Stop button or through a call to webc_BrowserStop.

HTML Browsers also contain a private data pointer that the user can set and retrieve to suit a particular application. For example, suppose one wishes to implement a dialog box for entering a user name and password. The Browser's private data pointer can be set to point to an instance of a structure that contains fields for user name and password. Then, callback functions can be bound to the individual input elements for the text boxes in the HTML document for the dialog box (see **Working with Custom Event Handlers**). These callbacks can retrieve the entered data from their respective elements via the WebC API, and save them in the private data pointer structure. Then, when the dialog box terminates, this structure will retain the user-entered information (see **Appendix B, Creating a Modal Dialog Box example**).

4.3.2 CREATE AND CONFIGURE AN HTML BROWSER

To create a new HTML Browser instance, the user must call webc_CreateBrowser (see Appendix A for a complete WebC API reference). This function takes two parameters: an initial configuration and a URL to display in the window (this URL can be reloaded at any time during the life of the HTML Browser by calling webc_BrowserHome). The initial configuration is specified as a pointer to an instance of the HTMLBrowserConfig structure.

The HTMLBrowserConfig structure is defined as follows:

```
struct HTMLBrowserConfig
{
    struct
    {
            void *parent_object;

            struct
            {
                    BCbool   fixed;
                    BCuint16 x;
                    BCuint16 y;
            } position;

            struct
            {
                    BCbool   fixed;
            } size;

            BCuint16 width;
            BCuint16 height;
            BCbool   minimized;
            BCbool   maximized;
            BCbool   closeable;
            BCbool   hidden;
            BCbool   hasframe;
            BCuint8  frame;

            struct
            {
                    BCbool  on;
                    BCbool  min_button;
                    BCbool  max_button;
                    BCbool  close_button;
            } title_bar;

            struct
            {
                    BCbool  on;
            } url_bar;

            struct
            {
                    BCbool  on;
            } button_bar;

            struct
            {
                    BCbool  on;
            } status_bar;

            struct
            {
                    BCbool  on;
            } load_animation;

            struct
            {
                    BCuint8  mode;
                    BCuint16 offset;
            } hscroll;

            struct
            {
                    BCuint8  mode;
                    BCuint16 offset;
            } vscroll;

            struct
            {
                    BCuint8  frame;
            } htmlwin;

    } window;

    RTP_PFUINT8 privatedata;
};
```

4.3.3 DESCRIPTION OF HTMLBROWSERCONFIG USER-CONFIGURABLE VALUES

The list of user configurable values is derived from the above struct definition.

WINDOW.PARENT_OBJECT

This is a pointer to the PegThing object to which to add the new browser window as a child object. If this is NULL (the default), then the browser window will be added to the PegPresentationManager (the root-level display).

WINDOW.POSITION.FIXED

Boolean value; if set to HTML_TRUE, then the browser window will not be user-moveable via the title bar.

WINDOW.POSITION.X
16 bit integer value; the x-coordinate in pixels of the left border of the browser window from the left edge of the screen.

WINDOW.POSITION.Y
16 bit integer value; the y-coordinate in pixels of the top border of the browser window from the top edge of the screen.

WINDOW.SIZE.FIXED
Boolean value; if set to HTML_TRUE, then the browser window will not be user-resizable.

WINDOW.WIDTH
16 bit integer value; the width in pixels of the browser window.

WINDOW.HEIGHT
16 bit integer value; the height in pixels of the browser window.

WINDOW.MINIMIZED
Boolean value; if set to HTML_TRUE, then the browser window will display as "minimized" (as an icon on the background of the window manager) until the window's icon is clicked or this value is set back to HTML_FALSE.

WINDOW.MAXIMIZED
Boolean value; if set to HTML_TRUE, then the browser window will display as "maximized" (taking up the full area of the display device).

WINDOW.CLOSEABLE
Boolean value; if set to HTML_FALSE, then the browser window will not respond to the user clicking the "close" button on the title bar.

WINDOW.HIDDEN
Boolean value; if set to HTML_TRUE, then the browser window will be removed from the display manager and so will not be visible.

WINDOW.FRAME
Integer value; controls the frame style of the browser window. Set to one of the following:

WEBC_FRAME_DEFAULT	-	The default frame style
WEBC_FRAME_RECESSED	-	3D recessed frame
WEBC_FRAME_RAISED	-	3D raised frame
WEBC_FRAME_NONE	-	No frame
WEBC_FRAME_THICK	-	Thick 3D border
WEBC_FRAME_THIN	-	Thin, one pixel border (no shading)

WINDOW.TITLE_BAR.ON
Boolean value; if set to HTML_TRUE, the browser window will display a title bar along the top of the window.

WINDOW.TITLE_BAR.MIN_BUTTON
Boolean value; if set to HTML_TRUE, and window.title_bar.on is set to HTML_TRUE, then a "minimize" button will be displayed on the title bar.

WINDOW.TITLE_BAR.MAX_BUTTON
Boolean value; if set to HTML_TRUE, and window.title_bar.on is set to HTML_TRUE, then a "maximize" button will be displayed on the title bar.

WINDOW.TITLE_BAR.CLOSE_BUTTON
Boolean value; if set to HTML_TRUE, and window.title_bar.on is set to HTML_TRUE, then a "close" button will be displayed on the title bar.

WINDOW.URL_BAR.ON
Boolean value; if set to HTML_TRUE, then the browser window will display a text entry box for entering a URL to load.

WINDOW.BUTTON_BAR.ON
Boolean value; if set to HTML_TRUE, then the browser window will display a bar of graphical buttons for navigation functions ("Back," "Forward," "Stop," "Home," and "Reload").

WINDOW.STATUS_BAR.ON
Boolean value; if set to HTML_TRUE, then the browser window will display a status bar on the bottom of the window.

WINDOW.LOAD_ANIMATION.ON
Boolean value; not currently supported.

WINDOW.HSCROLL.MODE
Integer value; controls the mode of the horizontal scroll bar in the root-level HTML display area. Set to one of the following:

WEBC_SCROLL_MODE_OFF	-	Never display scroll bar
WEBC_SCROLL_MODE_ON	-	Always display scroll bar
WEBC_SCROLL_MODE_AUTO	-	Only display scroll bar when needed (default)

WINDOW.HSCROLL.POSITION
16 bit integer value; the position of the horizontal scroll bar in pixels.

WINDOW.HSCROLL.RANGE
16 bit integer value; the maximum valid value of window.hscroll.position.

WINDOW.VSCROLL.MODE
Integer value; controls the mode of the horizontal scroll bar in the root-level HTML display area. See window.hscroll.mode for a description of values for this parameter.

WINDOW.VSCROLL.POSITION
16 bit integer value; the position of the vertical scroll bar in pixels.

WINDOW.VSCROLL.RANGE
16 bit integer value; the maximum valid value of window.vscroll.position.

WINDOW.HTMLWIN.FRAME
Integer value; controls the frame style of the root level HTML display area. See window.frame for a description of values for this parameter.

PRIVATEDATA
A private data pointer (far pointer to unsigned char). This

value is not used by WebC, and is available to the user for application-specific purposes. It can be set through the HTMLBrowserConfig structure (useful for setting it to an initial value before any page has been loaded) or through webc_BrowserSetPrivateData and retrieved via webc_BrowserGetPrivateData.

WebC also allows the user to retrieve the current configuration and/or modify it after an HTML Browser has been created, through webc_BrowserGetConfig and webc_BrowserSetConfig. See Appendix B, **Web Browser creation and configuration example**.

4.3.4 USING MODAL DIALOG BOXES

Normally, when a WebC application calls webc_CreateBrowser, a new HTML Browser is created, configured, and its initial URL loaded, and then the execution thread picks up after the call and proceeds from there. In some cases, it is desirable to have the current execution thread block waiting for an HTML Browser to terminate. WebC provides this ability through webc_BrowserExecute. Once webc_BrowserExecute has been called on the handle of a given HTML Browser, the Browser is said to be executing modally. A Browser that is executing modally cannot subsequently be changed to execute non-modally.

There are two ways an HTML Browser can terminate:

1. webc_DestroyBrowser is called with the handle of the given Browser;

2. The user clicks the close button on the Browser window’s title bar.

If an HTML Browser terminates in the first way, then the status value (integer) passed in to webc_DestroyBrowser will be the value returned from webc_BrowserExecute. If an HTML Browser terminates in the second way, then webc_BrowserExecute will return 0.

When a Browser is executing modally, it will be the only HTML Browser to receive GUI events until it terminates. Therefore, modally executing Browsers will always appear in the foreground until they are terminated.

4.3.5 HOW TO LOAD A NEW URL

The currently displayed document in an HTML Browser can be changed at any time by calling webc_BrowserSetUrl with a new URL to load. When this call returns, if there is no error, the new document will have been completely loaded and displayed in the HTML display area of the Browser window.

4.3.6 HOW TO TRAVERSE THE URL HISTORY

HTML Browsers keep a history of URLs that they have loaded. This history can be traversed via webc_BrowserBack and webc_BrowserForward. Whenever a new URL is entered explicitly by the user through the URL bar or when webc_BrowserSetUrl is called, all entries in the history that come after the current one are erased. So, if the user navigates to a web page via a sequence of links, then presses “Back” a few times, then manually enters and loads a new URL, and then presses “Forward,” no page will load.

Like webc_BrowserSetUrl, webc_BrowserBack and webc_BrowserForward will automatically update the display window.

4.3.7 SET AND GET PRIVATE DATA PER BROWSER

WebC allows the user to set and retrieve a single private data pointer for each HTML Browser. WebC does not use this value internally, so its function is entirely user defined. This pointer can be set through the HTMLBrowserConfig structure and webc_BrowserSetConfig (and retrieved through webc_BrowserGetConfig), or via webc_BrowserSetPrivateData (and retrieved through webc_BrowserGetPrivateData). See the above section, **How to Create and Configure an HTML Browser** for details on the first method.

4.4 WORKING WITH HTML DOCUMENTS

4.4.1 INTRODUCTION

HTML Documents are the internal representation of an HTML document. They consist of at least one HTML Element, of type HTML_BODY_ELEMENT, which represents the <BODY> tag of the document, and acts as the root of the Document Tree. WebC provides methods for searching the Document Tree for a specific element, setting a document’s source URL, and refreshing the display. In addition, WebC provides a method for opening an HTML Document for writing, and writing HTML source piece by piece directly into the document. WebC also allows the user to clear an HTML Document by deleting its content.

Each HTML Browser contains at least one HTML Document object, which is the document displayed in the HTML display area of the Browser window. HTML Documents can also be nested, if the root HTML document uses the <FRAMESET> and <FRAME> tags.

Usually, the application programmer need only access HTML Documents in response to events (i.e. in a custom event handler callback), or in order to write HTML source into a document. HTML Documents are created and destroyed automatically by WebC.

HTML Documents are referred to in the WebC API by their HDOC_HANDLE.

4.4.2 HOW TO ACCESS THE ROOT HTML DOCUMENT OF AN HTML BROWSER

When webc_CreateBrowser is called, an HTML Document (called the Root Document) is created to hold HTML content in that Browser. The handle for this HTML Document can be retrieved via webc_BrowserGetDocument. Likewise, the handle of the HTML Browser than owns an HTML Document can be retrieved (if one knows the handle of the Document) via webc_DocGetBrowser.

4.4.3 WRITING HTML DIRECTLY INTO AN HTML DOCUMENT

To write HTML directly to an HTML Document object, the Document must first be opened via webc_DocOpen. This will ensure correct parsing of HTML through multiple calls to webc_DocWriteHtml.

Once the Document has been opened, HTML source can be parsed into the Document Tree via webc_DocWriteHtml. It is worth noting that special characters which are used by HTML need to be replaced by escape codes if the desired result is to display those characters explicitly. These characters include the less than ('<'), greater than ('>'), and non-breaking spaces (white space that will not be reduced automatically to a single space). In order to facilitate ease of use when parsing new HTML into a Document, WebC provides a function, webc_DocWriteString, which will perform escape code substitutions automatically before parsing the text it is given. This means that anything passed to webc_DocWriteString will display exactly as it is written in the source string. Note, however, that no HTML tags will be parsed correctly if they appear in the string argument to webc_DocWriteString (instead, they will just show up as they look in the HTML source code).

After all new HTML has been parsed into an HTML Document the user should call webc_DocClose. At this point, no visible change has been affected in the Browser window. To refresh the display to reflect the Document's new content, the user should call webc_DocRefresh.

4.4.4 CHANGE A DOCUMENT'S URL AND RELOAD

The source URL for an HTML Document can be changed at any time using webc_DocSetUrl. If the refresh option is not set to HTML_TRUE for this call, then the new document will not be loaded, nor the current contents of the HTML Document discarded until webc_DocRefresh is called.

Changing the URL of an HTML Document directly (i.e. not through webc_BrowserLoadUrl) will only modify the URL history if the given Document is the Root Document.

4.4.5 HOW TO REFRESH THE DISPLAY

In addition to its uses that have already been discussed, the display of an HTML Document can be refreshed at any time by calling webc_DocRefresh. This may be preferable to calling webc_ElementRefresh multiple times on a large number of elements that have been modified (see also **Working with HTML Elements**).

4.4.6 SEARCHING FOR AN HTML ELEMENT WITHIN AN HTML DOCUMENT

To find a specific element by id, name, type, and/or index use webc_DocFindElement (see Appendix B for examples).

4.5 WORKING WITH HTML ELEMENTS

4.5.1 INTRODUCTION

As mentioned before, HTML Elements are the nodes of the Document Tree structure used internally by WebC to store loaded document information. HTML Elements are also the targets of HTMLEvents (see **Working with Custom Event Handlers**). Each tag and text fragment in an HTML document will generate an HTML Element node in the Document Tree. When tags and text are nested within a tag in the source document, their HTML Elements are created as children of the HTML Element of the tag within which they appear.

WebC provides methods for retrieving and modifying most of the properties of HTML Elements, as well as for getting and searching through the children and siblings of a particular HTML Element. WebC allows specific control over when and where HTML Elements are drawn (although all positioning is done by default, unless overridden, by the WebC HTML formatting and rendering engine). In addition, WebC provides functions for retrieving data from input-type elements such as <INPUT> tags, <TEXTAREA> tags, and <SELECT> and <OPTION> tags.

4.5.2 RETRIEVE AND MODIFY ELEMENT PROPERTIES

Many common HTML Element properties are available to the application programmer directly through Get and Set functions.

In the case where modifying an HTML Element property will change how the element (or another element) is drawn on the screen, the Set functions for a given property have a boolean parameter called refresh. Unless this parameter is set to HTML_TRUE, the element will not be redrawn until webc_ElementRefresh is called.

In the case where modifying an HTML Element property will necessitate the loading of a URL, the refresh parameter in the Set function is used to control whether the load occurs immediately or the next time webc_ElementRefresh is called.

Thus, the application programmer can control when/how often URL loads and screen refreshes occur. (In most cases, a Set function which necessitates a URL load will also necessitate a screen refresh).

4.5.3 LIST OF PROPERTIES ACCESSIBLE THROUGH WEBC

ID

Document-unique identifier string.
Pertains to: All elements
Read-only: webc_ElementGetId.

TYPE

Element type (defined by enumerated type HTMLElementType).
Pertains to: All elements
Read-only: webc_ElementGetType

NAME

Element name; not necessarily unique.
Pertains to: All elements
Read/Write: webc_ElementGetName, webc_ElementSetName

VALUE

The current value of an input field.
Pertains to:
- HTML_EDIT_STR_ELEMENT
- HTML_EDITBOX_ELEMENT
- HTML_BUTTON_ELEMENT
- HTML_CHECKBOX_ELEMENT
- HTML_HIDDEN_INPUT_ELEMENT
- HTML_OPTION_ELEMENT
- HTML_SELECT_ELEMENT
- HTML_RADIO_BUTTON_ELEMENT

Read/Write: webc_ElementGetValue, webc_ElementSetValue

CHECKED

The element's VALUE if selected, NULL otherwise.
Pertains to:
- HTML_CHECKBOX_ELEMENT
- HTML_RADIO_BUTTON_ELEMENT

Read/Write: webc_ElementGetChecked, webc_ElementSetChecked

SRC

The URL of the element's image or document source file.

Pertains to:
- HTML_IMAGE_ELEMENT

HTML_FRAME_ELEMENT
Read/Write: webc_ElementGetSrc, webc_ElementSetSrc

COLOR
Foreground color of the element.
Pertains to: All elements
Read/Write: webc_ElementGetColor, webc_ElementSetColor

BGCOLOR
Background color of the element.
Pertains to: All elements
Read/Write: webc_ElementGetBgColor, webc_ElementSetBgColor

BGIMAGE
Url of the element's background bitmap. Pertains to:
HTML_BODY_ELEMENT
HTML_TABLE_ELEMENT
HTML_TABLE_CELL_ELEMENT
Read/Write: webc_ElementGetBgImage, webc_ElementSetBgImage

POSITION
The position (x,y) in pixels of the upper left corner of an element relative to the upper left corner of its parent.
Pertains to: All elements
Read/Write: webc_ElementGetPosition, webc_ElementSetPosition

WIDTH
Width of the element in pixels.
Pertains to: All elements
Read/Write: webc_ElementGetWidth, webc_ElementSetWidth

HEIGHT
Height of the element in pixels.
Pertains to: All elements
Read/Write: webc_ElementGetHeight, webc_ElementSetHeight

VISIBILITY
Whether the element is visible.
Pertains to: All elements
Read/Write: webc_ElementSetHidden, webc_ElementSetVisible

STYLE
Style sheet information for this element.
Pertains to: All elements
Write-only: webc_ElementSetStyle

Note: See Appendix A for a full description of all the referenced API functions above.

4.5.3 NAVIGATING THE DOCUMENT TREE

4.5.4 FUNCTIONS DEFINED BY WEBC FOR NAVIGATING THE HTML DOCUMENT TREE

WEBC_ELEMENTGETPARENT
Returns an element's immediate parent

WEBC_ELEMENTGETFIRSTCHILD
Returns an element's first child

WEBC_ELEMENTGETLASTCHILD
Returns an element's last child

WEBC_ELEMENTGETNEXTSIBLING
Returns an element's next sibling

WEBC_ELEMENTGETPREVSIBLING
Returns an element's previous sibling

In general, users should ONLY use these functions if they absolutely need to navigate through the document tree at the lowest level. While WebC's internal representation of the HTML document structure for the most part is faithful to the original source document's tag nesting structure, there are a few exceptions, which should be understood by the user before attempting to manipulate the tree directly.

4.5.5 DRAWING AN HTML ELEMENT

Any HTML Element can be redrawn at any time using webc_ElementRefresh.

4.5.6 POSITIONING AN HTML ELEMENT

Normally, HTML Elements are positioned within the HTML display area automatically by WebC's internal HTML formatting algorithm. If the user wishes to position an element manually, this formatting can be over-ridden by using webc_ElementSetPosition. This function will take the specified element out of the text flow of the HTML document (thus the elements appearing immediately before and after will subsequently be drawn with no intervening space), and set its position in pixels relative to the position of its parent (see webc_ElementGetParent) element (which is still subject to change if the document is reformatted, and the user has not called webc_ElementSetPosition on the parent element).

The default positioning algorithm can be restored for an individual element at any time by using webc_ElementSetInline.

4.5.7 CONTROLLING HTML ELEMENT'S VISIBILITY

The user can set any HTML element to not be displayed by calling webc_ElementSetHidden on that element. This will remove the element from the display and also remove any effects the presence of the element had on surrounding elements in the document (the next time webc_ElementRefresh is called, or if the 'refresh' parameter is set to HTML_TRUE).

An element can be made visible again by calling webc_ElementSetVisible.

4.5.8 RESIZING AN HTML ELEMENT

Some HTML elements can be resized using webc_ElementSetWidth and webc_ElementSetHeight. These functions only affect images, tables, and block-level elements (such as <div> tags), which have been removed from the text flow using webc_ElementSetPosition. See the comments in Positioning an HTML Element for more details.

4.5.9 RETRIEVING USER INPUT

WebC provides a set of functions for setting and retrieving the values of HTML <FORM> fields, thus providing a robust method for obtaining user input and displaying feedback within an application.

4.5.10 HTML ELEMENT TYPE VALUES FOR USER INPUT ELEMENTS

HTML_EDIT_STR_ELEMENT
Edit boxes (<INPUT type=text>)

HTML_EDITBOX_ELEMENT
Scrollable text areas (<TEXTAREA>)

HTML_SELECT_ELEMENT
Combo boxes (<SELECT>)

HTML_RADIO_BUTTON_ELEMENT
Radio buttons (<INPUT type=radio>)

HTML_CHECKBOX_ELEMENT
Checkboxes (<INPUT type=checkbox>)

HTML_BUTTON_ELEMENT
Submit buttons (<INPUT type=submit>)

HTML_IMAGE_ELEMENT
Image buttons (<INPUT type=image>)

The contents of text-based inputs, such as edit boxes (<INPUT type=text> tags) and scrollable text entry boxes (<TEXTAREA> tags) can be set and retrieved via webc_ElementGetValue and webc_ElementSetValue.

The value of the currently selected item in a combo box (<SELECT> tag) can be retrieved by calling webc_ElementGetValue on the handle for that combo box element. The index of the currently selected item can be modified with webc_ElementSetSelected.

The ‘checked’ off status of boolean-type fields (radio buttons and checkboxes) can be obtained by calling webc_ElementGetChecked: if this function returns non-NULL, then the element in question IS checked off, otherwise it is not currently checked. The application programmer can set the ‘checked’ status of a radio button or checkbox via webc_ElementSetChecked.

Radio buttons can be grouped into a mutually exclusive set of options by assigning the same name to each radio button in the group. Here is an example of the HTML code to do this, and a WebC API code fragment that determines which radio button within the group is selected:

HTML Source:

```
Favorite Fruit: <BR>

<INPUT type=radio name=group1 value=apples> Apples <BR>
<INPUT type=radio name=group1 value=oranges> Oranges <BR>
<INPUT type=radio name=group1 value=bananas> Bananas <BR>
<INPUT type=radio name=group1 value=other> Other <BR>
```

Webster C fragment:

```
// We start with the HDOC_HANDLE of the document
//(hdoc) containing the above text, and set the
// char * p_radio_val to the value of the
// selected radio button.
HELEMENT_HANDLE helem;
// Find the first radio button with the group name
helem = webc_DocFindElement(hdoc, 0, ”group1",
                        HTML_RADIO_BUTTON_ELEMENT, 0);

// Now cycle through the rest of them looking for
//one that is ’checked’
while (helem != NULL)
     {

// If webc_ElementGetChecked returns non NULL,
// then this one is selected

p_radio_val = webc_ElementGetChecked(helem);
     if (p_radio_val != NULL)
     {
             break;
     }

// Get the next button in the group
helem = webc_ElementNext(helem, 0, ”group1",
                        HTML_RADIO_BUTTON_ELEMENT, 0);
     }
```

4.6 WORKING WITH CUSTOM EVENT HANDLERS

4.6.1 INTRODUCTION

One of WebC’s most powerful features is the ability to assign to an individual tag a C function that is called whenever an event that affects that tag is triggered. Such functions are called Custom Event Handlers.

WebC’s Custom Event Handler system is straightforward, but there are a few details that the event handler function writer must keep in mind in order to ensure correct operation.

In order to bind a Custom Event Handler to an element (tag) in an HTML document, two steps must be performed. First, the function must be registered with WebC under a unique function name using webc_RegisterEventCallback. Then, the event handler can be bound to an element by setting a special attribute in the HTML source for that tag called eventhandler.

4.6.2 HOW TO WRITE A CUSTOM EVENT HANDLER

The format of a Custom Event Handler Function is:

```
enum HTMLEventStatus func(       HBROWSER_HANDLE
hbrowser,
HDOC_HANDLE hdoc,
HELEMENT_HANDLE helem,
struct HTMLEvent *pEvent,
char *pParam);
```

where hbrowser, hdoc, and helem are the HTML Browser, HTML Document, and HTML Element, respectively, to whom the event has happened. The type of event and any event-specific related data (for example, on a key down event, the key code that was pressed) are passed in the HTMLEvent structure. Finally, pParam is an optional function parameter string that is passed to a custom event handler in the following manner:

Suppose the user has registered an event handler called “LogToDisk,” which is bound to an HTML <INPUT> tag in the following way:

```
<INPUT type=submit eventhandler=”LogToDisk(Button
Pressed)”>
```

When an event happens to this HTML Element, the C function registered as “LogToDisk” will be called with a pParam argument of “Button Pressed.”

4.6.3 HTML EVENT TYPES

4.6.4 EVENT MESSAGE TYPES DEALT WITH BY WEBC.

Any of these may cause a call to a custom event handler. The description for each event type also includes under what circumstances and for which type of elements the event is triggered, along with any special data passed inside the HTMLEvent structure.

HTML_EVENT_CLICK
Single mouse click

Triggered whenever the mouse is clicked over an HTML Element that receives mouse input. These elements are: <BODY>, <A>, <INPUT>, <IMG>, <TEXTAREA>, <SELECT>/<OPTION>.

Event Data: data.position.x, data.position.y; the x and y coordinates of the mouse cursor when the event occurred.

HTML_EVENT_DBLCLICK
Double mouse click

Triggered whenever the mouse is double-clicked over an HTML Element that receives mouse input. See HTML_EVENT_CLICK for a list of these elements.

Event Data: data.position.x, data.position.y; the x and y coordinates of the mouse cursor when the event occurred.

HTML_EVENT_KEYDOWN
Key Pressed (but not Released)

Triggered whenever a key is pressed down. This event is sent to the HTML Element that holds the current input focus. Any HTML Element can receive input focus, but only those that receive mouse input (see HTML_EVENT_CLICK for a list of these elements) will automatically receive input focus in response to mouse input.

Event Data: data.key; 16-bit scan code of the key pressed. 0-127 are ASCII-equivalent; special keys are represented using the PEG key scan code conventions (see the PEG manual for more details).

HTML_EVENT_KEYPRESS
Key Pressed and Released

Triggered whenever a key is pressed. This event is sent to the HTML Element that holds the current input focus. Any HTML Element can receive input focus, but only those that receive mouse input (see HTML_EVENT_CLICK for a list of these elements) will automatically receive input focus in response to mouse input.

Event Data: data.key; 16-bit scan code of the key pressed. 0-127 are ASCII-equivalent; special keys are represented using the PEG key scan code conventions (see the PEG manual for more details).

HTML_EVENT_KEYUP
Key Released

Triggered whenever a key that was previously pressed is released. This event is sent to the HTML Element that holds the current input focus. Any HTML Element can receive input focus, but only those that receive mouse input (see HTML_EVENT_CLICK for a list of these elements) will automatically receive input focus in response to mouse input.

Event Data: data.key; 16-bit scan code of the key pressed. 0-127 are ASCII-equivalent; special keys are represented using the PEG key scan code conventions (see the PEG manual for more details).

HTML_EVENT_MOUSEDOWN
Mouse Button Pressed (but not Released)

Triggered when the mouse button is pressed over an HTML Element that receives mouse input. See HTML_EVENT_CLICK for a list of these elements.

Event Data: data.position.x, data.position.y; the x and y coordinates of the mouse cursor when the event occurred.

HTML_EVENT_MOUSEMOVE
Mouse Cursor Moved

Triggered when the mouse button is moved over an HTML Element that receives mouse input. See HTML_EVENT_CLICK for a list of these elements.

Event Data: data.position.x, data.position.y; the new x and y coordinates of the mouse cursor.

HTML_EVENT_MOUSEOUT
Mouse Cursor has left an Element

Triggered when the mouse cursor exits the bounding rectangle of an HTML Element that receives mouse input. See HTML_EVENT_CLICK for a list of these elements.

Event Data: none.

HTML_EVENT_MOUSEOVER
Mouse Cursor has entered an Element

Triggered when the mouse cursor enters the bounding rectangle of an HTML Element that receives mouse input. See HTML_EVENT_CLICK for a list of these elements.

Event Data: data.position.x, data.position.y; the x and y coordinates of the mouse cursor when the event occurred.

HTML_EVENT_MOUSEUP
Mouse Button Released

Triggered when the mouse button is released over an HTML Element that receives mouse input. See HTML_EVENT_CLICK for a list of these elements.

Event Data: data.position.x, data.position.y; the x and y coordinates of the mouse cursor when the event occurred.

HTML_EVENT_FOCUS
Element has been given the input focus

Triggered when an Element receives the input focus as a result of mouse input or an explicit call to webc_TriggerEvent. When an Element has been given the input focus, all subsequent key messages will be passed to that Element.

Event Data: none.

HTML_EVENT_UNFOCUS
Element has lost the input focus

Triggered when an Element loses input focus as a result of mouse input or an explicit call to webc_TriggerEvent. The input focus will be given to the Element that had the input focus when the recipient of HTML_EVENT_UNFOCUS originally received the input focus.

Event Data: none.

HTML_EVENT_LOAD
Element has completed loading

Triggered on HTML Elements that initiate a URL load; this event is sent when the URL that this HTML Element initiated, as well as all URL loads that were initiated as a result of the first one, has finished loading. For example, HTML_EVENT_LOAD is sent to the <BODY> Element when a document has completely finished loading.

Event Data: none.

HTML_EVENT_UNLOAD
Element is about to be purged from memory

Triggered right before an Element is deleted. Allows cleanup operations to take place.

Event Data: none.

HTML_EVENT_SUBMIT
Form submission

Triggered only for <FORM> Elements when a submit control is activated. A form submission can be forced by using webc_TriggerEvent to pass an HTML_EVENT_SUBMIT event to a <FORM> Element.

Event Data: data.elem; the handle of the Element that triggered the submission.

HTML_EVENT_CHILDUPDATE
A child element's appearance has been modified

Triggered whenever an Element's size or float property has been modified, necessitating a re-format of the parent element (the target of this event). This event is usually sent internally from a child to its parent, and should probably not be sent or handled by the application programmer.

Event Data: data.elem; the handle of the Element that was updated.

HTML_EVENT_CHANGE
The Value property of an Element has been modified

Triggered when an Element's Value property has changed. Only relevant for Elements that have a Value property (see **Working with HTML Elements** for a list of Element properties). This event is sent less frequently (if logical) than HTML_EVENT_CHANGE; for example, it is only sent to a text box when the user types enter or moves to a different form field.

Event Data: none.

HTML_EVENT_EDIT
The Value property of an Element has been edited

Triggered when an Element's Value property has changed. Only relevant for Elements that have a Value property (see **Working with HTML Elements** for a list of Element properties). This event is sent less frequently (if logical) than HTML_EVENT_CHANGE; for example, it is only sent to a text box when the user types enter or moves to a different form field.

Event Data: none.

4.6.5 EVENT HANDLER RETURN VALUE

Custom Event Handler functions in WebC must return a value from the enumeration HTMLEventStatus. The three values that are acceptable are:

HTML_EVENT_STATUS_DONE
HTML_EVENT_STATUS_CONTINUE
HTML_EVENT_STATUS_HALT

The return value of an Event Handler will determine whether WebC will continue to look for an event handler (either JavaScript or the default handler) to handle this event, and whether or not it should immediately return without further processing.

In a Custom Event Handler, if the event is processed successfully, and the desired behavior is for WebC to not perform the default handling as well, then the function should return HTML_EVENT_STATUS_DONE. If the event was not processed successfully, or the user does wish WebC to perform default handling in addition, then the function should return HTML_EVENT_STATUS_CONTINUE. If, however, the Document Tree was deleted as a result of the processing that occurred in a custom event handler (for example, if the handler loads a new URL or calls webc_DestroyBrowser on the current HTML Browser), then the event handler MUST return HTML_EVENT_STATUS_HALT. If it does not return this value in this case, it will almost certainly cause an error in program execution.

4.6.6 A CUSTOM EVENT HANDLER EXAMPLE

For example, suppose the user wishes to design a dialog box with (among other things) a button which will initiate some process in the embedded system WebC is running on. First, as part of the WebC initialization procedure, after webc_Init is called, the application must call webc_RegisterEventCallback with the C name of the function, and a name that will be used to refer to that function in HTML documents. Suppose that the function's registered name is "doSomething." Then, the HTML for the button in the dialog box document might look something like:

```
<INPUT type=submit value='Start'
eventhandler='doSomething'>
```

This will produce a button labeled “Start” that will call the C function that was registered in our application whenever an event happens to that button element.

In this case, most events that can occur are probably of no interest to the custom event handler; we only care about a mouse click event (or maybe a key event with the return key). Thus the event handler function must first check the event type to make sure it is a click event.

4.6.7 WEBC EVENT HANDLING ALGORITHM

When an event is triggered, WebC performs an algorithm to find one or more event handlers to process it. This algorithm is described here, so that the user of WebC will understand how custom event handlers, JavaScript event handlers, and the default event handler for an Element type are used and coordinated.

The first way WebC will try to handle an incoming event is by searching for a Custom Event Handler for the Element. If it can find such an event handler, it will call it, and depending on its return value, continue to search for JavaScript and/or default handlers. If no Custom Event Handler is found for a particular element, then WebC will move up to the Element’s parent and check for a Custom Event Handler on that Element, and so forth on up the Document Tree, until either a Custom Event Handler is found, or WebC reaches the root of the tree and finds nothing.

If either no Custom Event Handler was found, or one was found and called, but it returned HTML_EVENT_STATUS_CONTINUE, then WebC will look for a JavaScript event handler (provided JavaScript is turned on; if not, this part of the algorithm will be skipped). Similar to the Custom Event Handler search method, WebC will search up the Document Tree until it finds a JavaScript handler to run, or reaches the top of the tree.

If WebC finds a JavaScript event handler and executes it, and it returns true, or no JavaScript event handler is found for the event (or JavaScript is not enabled), then WebC will execute the default event handler for the type of Element that received the event.

4.6.8 TRIGGERING HTML EVENTS EXPLICITLY

The application programmer can also send an event to an Element directly, via webc_TriggerEvent. Events triggered in this way are handled exactly the same as events that occur as a part of normal, internal WebC operation.

4.6.9 PREDEFINED CUSTOM EVENT HANDLERS

There is one custom event handler that is built in to WebC, called “closeWindow.” It takes one parameter, the return value to pass to webc_DestroyBrowser, and destroys the current HTML Browser when it receives a click event.

4.7 PROVIDING CUSTOM HANDLING FOR META TAGS

4.7.1 INTRODUCTION

WebC provides the ability to assign C callback functions to HTML <META> tags based on the name attribute of the tag. The return value of the callback will determine whether or not default <META> tag processing will occur for the given tag.

Custom META tag handling is not performed until the entire document has been loaded, and is performed on META tags in document order.

4.7.2 WRITING A CUSTOM META TAG HANDLER

The following function prototype is used to define a custom META tag handler:

```
HTMLMetaStatus MetaCallback(char *pContent,
    HDOC_HANDLE hdoc);
```

where pContent is a pointer to a null-terminated string equal to the value of the content attribute of the <META> tag, and hdoc is the handle of the HTML Document within which the tag occurred.

As noted before, the return value of a META tag handler callback determines whether default processing will occur on this tag. If the callback returns HTML_META_STATUS_CONTINUE, then default processing **will** occur; if it returns HTML_META_STATUS_STOP, then default processing **will not** occur.

If a META callback causes the given HTML Document to reload or be deleted, then the META callback **must** return HTML_META_STATUS_HALT.

4.7.3 REGISTERING A CUSTOM META TAG HANDLER

Before WebC will recognize and process custom META tag types, an event handler must be registered for each custom type which is to be processed. This is done via webc_RegisterMetaCallback (see Appendix A).

See Appendix B for an example of a Custom META Tag Handler.

4.8 DEFINING URL MACROS

4.8.1 INTRODUCTION

WebC provides the ability to define text-substitution macros that are replaced in URLs when they are parsed to load a resource from a particular location. For example, if the following HTML code fragment appears in a document:

```
<A href="http://%serverip%/page.html">
```

and the application programmer has defined a URL macro called %serverip% that maps to “192.168.0.1,” then when the user clicks on the given link, the URL:

```
http://192.168.0.1/page.html
```

will be loaded.

4.8.2 URL MACRO SYNTAX

URL Macros in HTML documents use MS-DOS batch file variable substitution syntax; i.e., %<macro name>% where <macro name> is the name of the macro to invoke. Furthermore, a string argument may optionally be passed in to a macro:

```
%<macro name>(<argument string>)%
```

where <argument string> is a string that will be passed along to the URL Macro callback function used to perform the substitution.

4.8.3 HOW TO DEFINE A MACRO

URL Macros are defined in WebC by registering a callback function under a unique macro name. The prototype for a URL Macro callback is:

```
int UrlMacro(char *param, char *replace);
```

where param is the argument string described above, and replace is a pointer to a buffer into which the callback should write its replacement. A URL Macro callback function must return the number of characters (not including a null terminator) written to replace as a result of expanding the macro.

To register a URL Macro, use webc_RegisterUrlMacro (see Appendix B for an example).

4.9 USING WEBC VIRTUAL FILES

4.9.1 INTRODUCTION

In addition to the standard URL transport protocols, HTTP, FILE, HTTPS, JAVASCRIPT, etc., WebC also defines a protocol for accessing pre-compiled data files. This protocol is called WEBC. WebC manages an internal table of file names matched with a buffer, length, and MIME type which can be accessed through this protocol. These files are called Virtual Files.

For example, suppose there is an entry in the table named "index.html", which points to a buffer in memory where the string "<HTML>Hello, World.</HTML>" resides. If the URL "webc://index.html" is typed into the URL bar, the above HTML document will be loaded, parsed, and displayed, resulting in **'Hello, World'** being displayed in the HTML display area of the current HTML Browser window. WebC Virtual Files can be very useful in systems where there is no local file system.

4.9.2 HOW TO CREATE A VIRTUAL FILE

New virtual files can be added to the virtual table using webc_CreateVFile. This function takes the file name (everything that comes after the "webc://", a buffer containing the file contents, the size in bytes of this buffer, and the MIME type of the data contained in the file. It is essential for correct WebC operation that the MIME type field be correctly set. Otherwise, HTML data will not be parsed and rendered as HTML, images will not be processed correctly, and so forth.

Virtual files can be created before any HTML Browsers are created.

4.9.3 HOW TO DELETE A VIRTUAL FILE

Virtual Files can also be deleted from the table if they are no longer needed by an application by calling webc_DeleteVFile on the virtual file name.

4.9.4 USING VIRTUAL FILES

After a virtual file has been created, it can be used in HTML the same as an HTTP or FILE URL. Virtual files can also reference other virtual files; note that if the protocol is not specified in a URL that occurs within a virtual file, WebC will assume it is a relative URL and attempt to load that file from the virtual table.

Directory names can also be included in virtual file names as they are created. So, if the application programmer creates a virtual file named "images/button1.gif", then it can be accessed with the URL "webc://images/button1.gif".

4.10 USING WEBC TO MANAGE COOKIES

4.10.1 INTRODUCTION

WebC provides three functions, webc_CreateCookie, webc_FindCookie, and webc_DestroyCookie for managing HTTP cookies. The cookies created and accessed by these functions are exactly the same as the cookies set and retrieved via HTTP.

4.10.2 WHAT IS A COOKIE?

A cookie is a name/value string pair that is set when a document is loaded from a server over HTTP. A cookie also has a path and host name associated with it. Whenever an HTTP request is made, the HTTP client scans through its list of cookies that have been set from previous HTTP responses for any that match the host and path that the new request is being made to. All cookies that match the host and path (a host is a match if the domain matches, a path if the cookie path is a prefix of the requested path) are sent as name/value pairs along with the request.

Cookies provide a way for HTTP clients to communicate state information to HTTP servers. They also provide a mechanism for servers to store such state information on the client-side rather than maintaining a huge database of all the IP addresses of every client that makes requests of the server.

4.10.3 ADDING A NEW COOKIE

The user can add new cookies manually on the client side by calling webc_CreateCookie. Any cookies added in this way will be sent to their appropriate servers as described above.

4.10.4 SEARCHING FOR AN EXISTING COOKIE

The user can search through the list of cookies on the client using webc_FindCookie. This function allows the user to search by name, value, host, path, and/or index (any combination thereof). Note: there is no distinction made in WebC between user-created cookies and server-created cookies. The webc_FindCookie function will return a handle to the cookie it found, or NULL if no matching cookie could be located.

4.10.5 DELETING A COOKIE

A cookie can be deleted at any time by calling webc_DestroyCookie on the handle of the cookie to be deleted (the return value of webc_FindCookie).

4.10.6 ACCESSING THE CONTENTS OF A COOKIE

The user can retrieve the name, value, host, and/or path of a cookie from its handle by using webc_CookieGetName, webc_CookieGetValue, webc_CookieGetHost, and webc_CookieGetPath respectively.

The values returned by these functions must not be modified in any way. They must be treated as read-only. Once a cookie is set, the only way to change its value is to call webc_CreateCookie with the same name, path, and host, but a different value.

4.11 DEFINING NEW JAVASCRIPT FUNCTIONS FROM C

4.11.1 INTRODUCTION

WebC allows the user to create native C functions that can be called in JavaScript code. There are only two API functions used for the JavaScript interface, webc_JScriptDefineFunction and webc_JScriptGetDocument.

The JavaScript interface defined by WebC uses the SpiderMonkey API. More information about the SpiderMonkey API can be found at www.mozilla.org/js.

4.11.2 ADDING A NEW JAVASCRIPT FUNCTION

To define a JavaScript function the user must create a function with the following signature JSBool someName(JSContext *cx, JSObject *obj, uintN argc, jsval *argv, jsval *rval); The cx parameter should be used to get a handle to the document that the function was called from via webc_JScriptGetDoc(cx). The parameter obj is a pointer to the global object structure and should not be needed by the user. The argc parameter is the number of arguments that the function has been given. The argv parameter is the parameter list. Always check to see if argc is at least the size of the arguments needed before accessing argv. The final parameter, rval, should be set to the desired return value of the JavaScript function. If the JavaScript function has no return value then this value should be set to JSVAL_VOID.

The native C function should return either JS_TRUE or JS_FALSE. If JS_FALSE is returned, then the JavaScript interpreter will halt and no further JavaScript execution will occur within that script.

The user should call webc_JScriptDefineFunction after at least one browser window has been created. The function will then live in all browser windows for the life of the program. That allows multiple windows and frames to use the same function without having to call webc_JScriptDefineFunction multiple times.

4.11.3 CREATING FUNCTIONALITY

All values in JavaScript are typeless. The user must use SpiderMonkey defined functions to convert typeless JavaScript values to and from typed C values. The following C function moves the current browser window to a location specified by JavaScript.

```
JSBool moveBrowser(JSContext *cx, JSObject *obj, uintN
argc, jsval *argv, jsval *rval)
{
HDOC_HANDLE hdoc;
HBROWSER_HANDLE hbrowser;
jsdouble x,y = 0;
struct HTMLBrowserConfig config;

//first check to see if we have enough arguments
if (argc > 1)
{
     //Get the document handle
     hdoc = webc_JScriptGetDoc(cx);
     hbrowser = webc_DocGetBrowser(hdoc);
     //change the jsvals to numbers (and make sure they
can be changed to numbers)
     if ( JS_ValueToNumber(cx, argv[0], &x) == JS_TRUE &&
      JS_ValueToNumber(cx, argv[0], &y) == JS_TRUE)
     {
               webc_BrowserGetConfig(hbrowser, &config);
               //update the config
               config.window.position.x = (int)x;
               config.window.position.y = (int)y;
               webc_BrowserSetConfig(hbrowser, &config);
     }
}
*rval = JSVAL_VOID;
return JS_TRUE;
}
```

The JavaScript source code to call this function whenever someone clicks on the browser is <body onClick="moveBrowser(200, 150);">. This assumes that the name parameter passed into webc_JScriptDefineFunction was moveBrowser.

Functions like JS_ValueToNumber are defined by the SpiderMonkey API. If this function returned an integer value, the code would look like: *rval=INT_TO_JSVAL(x);.

There also functions and macros for other types, such as Objects, Strings, integers, and functions. The full list can be found at http://www.mozilla.org/js/spidermonkey/.

CHAPTER 5

WEBC API FUNCTIONS

WEBC_BROWSERADDTOHISTORY

Function:

Add a location to the HTML Browser history.

Summary:

#include “webc.h”

webc_BrowserAddToHistory(**HBROWSER_HANDLE** browser, char* url_str);

browser	- The HTML Browser handle
url_str	- The URL to put in the history (a copy of the string will be created)

Description:

Can be used to directly modify the URL history.

Returns:

0 on success, otherwise -1

See Also:

webc_BrowserRefresh(), webc_BrowserBack(), webc_BrowserForward()

WEBC_BROWSERBACK

Function:

Load the previous location in the history.

Summary:

#include “webc.h”

int webc_BrowserBack(**HBROWSER_HANDLE** browser);

browser - The handle of the HTML Browser

Description:

This function is functionally identical to pressing the “Back” button in an HTML Browser window.

Returns:

0 on success, otherwise -1

See Also:

webc_BrowserForward(), webc_BrowserHome(), webc_BrowserRefresh(), webc_BrowserLoadUrl()

WEBC_BROWSEREXECUTE

Function:

Wait for an HTML Browser to terminate.

Summary:

#include “webc.h”

int webc_BrowserExecute(**HBROWSER_HANDLE** browser);

browser - The handle of the HTML Browser

Description:

This function will block until the given HTML Browser terminates by either:

1. The user closing the window.
2. JavaScript closes the window.
3. The window is closed by a call to **webc_DestroyBrowser()** inside a custom event handler.

Returns:

The value passed into **webc_DestoryBrowser()**

See Also:

webc_DestoryBrowser()

WEBC_BROWSERFOCUS

Function:

Set the active browser window.

Summary:

#include “webc.h”

int webc_BrowserFocus(**HBROWSER_HANDLE** browser);

browser - The handle of the HTML Browser

Description:

Moves the input focus to the window of the given HTML Browser handle and brings that window to the front of other windows.

Returns:

0 on success, otherwise -1

See Also:

HBROWSER_HANDLE, webc_ElementSetFocus()

WEBC_BROWSERFORWARD

Function:

Load the next location in the history.

Summary:

#include "webc.h"

int webc_BrowserForward(**HBROWSER_HANDLE** browser);

browser - The handle of the HTML Browser

Description:

This function is functionally identical to pressing the "Forward" button in an HTML Browser window.

Returns:

0 on success, otherwise -1

See Also:

webc_BrowserBack(), webc_BrowserHome(), webc_BrowserRefresh(), webc_BrowserLoadUrl()

WEBC_BROWSERGETCONFIG

Function:

Retrieve the current browser configuration.

Summary:

#include "webc.h"

int webc_BrowserGetConfig (**HBROWSER_HANDLE** browser, struct HTMLBrowserConfig* config);

browser	- The handle of the HTML Browser
config	- Pointer to a pre-allocated instance of struct HTMLBrowserConfig to fill with the current configuration.

Description:

This function can be used to obtain information about the current configuration of an HTML Browser. The structure that is set by this function can also be modified and passed to **webc_BrowserSetConfig()** to change the current configuration.

Returns:

0 on success, otherwise -1

See Also:

webc_BrowserSetConfig(), HTMLBrowserConfig()

WEBC_BROWSERGETDOCUMENT

Function:

Get an HTML browser's root document handle.

Summary:

#include "webc.h"

HDOC_HANDLE webc_BrowserGetDocument (**HBROWSER_HANDLE** browser);

browser	- The handle of the HTML Browser
config	- Pointer to a pre-allocated instance of struct HTMLBrowserConfig to fill with the current configuration.

Description:

Need description

Returns:

The handle of the root document for this HTML browser

See Also:

WEBC_BROWSERGETPRIVATEDATA

Function:

Retrieve the user-specified data pointer associated with an HTML Browser.

Summary:

#include "webc.h"

RTP_UINT8* webc_BrowserGetPrivateData **(**HBROWSER_HANDLE browser);

browser - The handle of the HTML Browser

Description:

Need description

Returns:

The pointer passed into **webc_BrowserSetPrivateData()** if there is one, otherwise **NULL**.

See Also:

webc_BrowserSetPrivateData()

WEBC_BROWSERHOME

Function:

Reload the home page.

Summary:

#include “webc.h”

int webc_BrowserHome(**HBROWSER_HANDLE** browser);

browser - The handle of the HTML Browser

Description:

This function is functionally identical to pressing the “Home” button in an HTML Browser window.

Returns:

0 on success, otherwise -1

See Also:

webc_BrowserForward(), webc_BrowserBack(), webc_BrowserRefresh(), webc_BrowserLoadUrl()

WEBC_BROWSERLASTURL

Function:

Get the last url loaded.

Summary:

#include “webc.h”

RTP_PFCHAR
webc_BrowserLastUrl(**HBROWSER_HANDLE** browser);

browser - The handle of the HTML Browser

Description:

Returns a pointer to the last url that was successfully loaded.

Returns:

The last successful URL

See Also:

webc_BrowserLoadUrl()

WEBC_BROWSERLOADURL

Function:

Load a location.

Summary:

#include “webc.h”

int webc_BrowserLoadUrl(**HBROWSER_HANDLE** browser, char* url_str);

browser - The handle of the HTML Browser
url_str - The URL to load

Description:

This function is functionally identical to entering a string in the URL bar of an HTML Browser window.

Returns:

0 on success, otherwise -1

See Also:

webc_BrowserForward(), webc_BrowserHome(), webc_BrowserRefresh(), webc_BrowserBack()

WEBC_BROWSERREFRESH

Function:

Reload the current page.

Summary:

#include “webc.h”

int webc_BrowserRefresh(**HBROWSER_HANDLE** browser);

browser - The handle of the HTML Browser
url_str - The URL to load

Description:

This function is functionally identical to pressing the “Refresh” button in the HTML Browser window.

Returns:

0 on success, otherwise -1

See Also:

webc_BrowserForward(), webc_BrowserHome(), webc_BrowserRefresh(), webc_BrowserBack(), webc_BrowserLoadUrl()

WEBC_BROWSERRESTART

Function:

Free a browser and all of its resources, and create a new browser with the same configuration.

Summary:

#include “webc.h”

void webc_BrowserRestart(**HBROWSER_HANDLE** browser);

browser - The handle of the HTML Browser to restart

Description:

This function is intended to exist as a means to quickly free memory.

Returns:

Nothing

See Also:

webc_CreateBrowser(), webc_DestroyBrowser()

WEBC_BROWSERSETCONFIG

Function:

Set the browser window configuration.

Summary:

#include “webc.h”

int webc_BrowserSetConfig(**HBROWSER_HANDLE** browser, struct HTMLBrowserConfig* config);

browser - The HTML Browser to configure
config - The new configuration

Description:

This function is used to change the appearance of the HTML Browser window and also to associate a private data pointer to an HTML Browser handle. (This can be useful when using custom event handlers).

Returns:

0 on success, otherwise -1

See Also:

webc_CreateBrowser(), webc_BrowserGetConfig(), HTMLBrowserConfig()

See Section 4.3 for a discussion of the browser configuration

WEBC_BROWSERSETPRIVATEDATA

Function:

Associate a data pointer with an HTML Browser.

Summary:

#include "webc.h"

void webc_BrowserSetPrivateData(**HBROWSER_HANDLE** browser, RTP_UINT8* data);

browser	- The HTML Browser handle
data	- The data pointer to associate

Description:

The data passed into this function is not used by WebC . The purpose of this routine is to associate application-specific data on a per-HTML Browser basis. This data can be retrieved at any time using **webc_BrowserGetPrivateData()**.

Returns:

Nothing

See Also:

webc_BrowserGetPrivateData()

WEBC_BROWSERSTOP

Function:

Abort all pending load jobs in an HTML Browser.

Summary:

#include "webc.h"

int webc_BrowserStop(**HBROWSER_HANDLE** browser);

browser - The handle of the HTML Browser

Description:

This function stops an in-progress load by cancelling all outstanding jobs in the HTML Browser's load queue. It is functionally identical to pressing the "Stop" button in the HTML Browser window.

Returns:

0 on success, otherwise -1

See Also:

webc_BrowserForward(), webc_BrowserHome(), webc_BrowserRefresh(), webc_BrowserBack(), webc_BrowserLoadUrl()

WEBC_COOKIEGETHOST

Function:

Get the host that a cookie was received from.

Summary:

#include “webc.h”

char* webc_CookieGetHost(**COOKIE_HANDLE** cookie);

cookie - Handle of the cookie

Description:

Retrieve a cookie’s value.

Returns:

The cookie’s host

See Also:

webc_CookieGetName(), webc_CookieGetValue(), webc_CookieGetPath()

WEBC_COOKIEGETNAME

Function:

Retrieve the name of a cookie.

Summary:

#include “webc.h”

char* webc_CookieGetName(**COOKIE_HANDLE** cookie);

cookie - Handle of the cookie

Description:

Retrieve the name of a cookie.

Returns:

The name of the cookie

See Also:

webc_CookieGetValue(), webc_CookieGetHost(), webc_CookieGetPath()

WEBC_COOKIEGETPATH

Function:

Get the path of a cookie.

Summary:

#include “webc.h”

char* webc_CookieGetPath(**COOKIE_HANDLE** cookie);

cookie - Handle of the cookie

Description:

Get the path of a cookie.

Returns:

The cookie’s path

See Also:

webc_CookieGetName(), webc_CookieGetValue(), webc_CookieGetHost()

WEBC_COOKIEGETVALUE

Function:

Retrieve a cookie’s value.

Summary:

#include “webc.h”

char* webc_CookieGetValue(**COOKIE_HANDLE** cookie);

cookie - Handle of the cookie

Description:

Retrieve a cookie’s value.

Returns:

The cookie’s value

See Also:

webc_CookieGetName(), webc_CookieGetHost(), webc_CookieGetPath()

WEBC_COOKIESETHOST

Function:

Set a cookie's host.

Summary:

#include "webc.h"

void webc_CookieSetHost(**COOKIE_HANDLE** cookie, char* host);

cookie	- Handle of the cookie
host	- Pointer to the string containing the cookie's new host

Description:

Set a cookie's host.

Returns:

Nothing

See Also:

webc_CookieSetName(), webc_CookieSetValue(), webc_CookieSetPath()

WEBC_COOKIESETNAME

Function:

Set the name of a cookie.

Summary:

#include "webc.h"

void webc_CookieSetName(**COOKIE_HANDLE** cookie, char* name);

cookie	- Handle of the cookie
name	- Pointer to the string containing the cookie's new name

Description:

Set the name of a cookie.

Returns:

Nothing

See Also:

webc_CookieSetValue(), webc_CookieSetHost(), webc_CookieSetPath()

WEBC_COOKIESETPATH

Function:

Set the path of a cookie.

Summary:

#include “webc.h”

void webc_CookieSetPath(**COOKIE_HANDLE** cookie, char* path);

cookie	- Handle of the cookie
path	- Pointer to the string containing the cookie’s new path

Description:

Set the path of a cookie.

Returns:

Nothing

See Also:

webc_CookieSetName(), webc_CookieSetValue(), webc_CookieSetHost()

WEBC_COOKIESETVALUE

Function:

Set the value of a cookie.

Summary:

#include “webc.h”

void webc_CookieSetValue(**COOKIE_HANDLE** cookie, char* value);

cookie	- Handle of the cookie
value	- Pointer to the string containing the cookie’s new value

Description:

Set the value of a cookie.

Returns:

Nothing

See Also:

webc_CookieSetName(), webc_CookieSetHost(), webc_CookieSetPath()

WEBC_CREATEBROWSER

Function:

Create a new HTML Browser window.

Summary:

#include “webc.h”

HBROWSER_HANDLE webc_CreateBrowser(struct HTMLBrowserConfig* config, char* home);

config	- The initial configuration of the new browser window
home	- URL of the initial page to load in this window. The browser will also return to this page when its ‘Home’ button (if it is configured to have one) is pressed, or when **webc_BrowserHome()** is called.

Description:

Allocates resources for and displays a new web browser window with the specified home page document.

Returns:

A handle to the newly created HTML Browser if successful, NULL otherwise

See Also:

webc_DestroyBrowser(), HTMLBrowserConfig()

WEBC_CREATECOOKIE

Function:

Create a cookie with the specified parameters.

Summary:

#include “webc.h”

COOKIE_HANDLE webc_CreateCookie(char* name, char* value, char* host, char* path);

name	- The name of the cookie
value	- Value the cookie will store
host	- Domain the cookie was received from
path	- Path of the cookie

Description:

Given a cookie name, its value, host name and path, this function will return a handle to the newly created cookie.

Returns:

COOKIE_HANDLE: Handle to the new cookie

See Also:

webc_DestroyCookie()

WEBC_DESTROYBROWSER

Function:

Close an HTML Browser window and free all its resources.

Summary:

#include “webc.h”

void webc_DestroyBrowser(**HBROWSER_HANDLE** browser, int status);

browser	- The handle of the HTML Browser to close
status	- If webc_BrowserExecute has been called on this browser, then this parameter will determine the return value of webc_BrowserExecute(). This is useful for simple dialog boxes which ask the user to choose between several choices. (see How to use an HTML Browser as a dialog box)

Description:

Call this to manually close an HTML Browser. **DO NOT JUST CALL** delete!

Returns:

Nothing

See Also:

webc_CreateBrowser(), webc_BrowserExecute()

WEBC_DESTROYCOOKIE

Function:

Destroys a cookie previously created with webc_CreateCookie.

Summary:

#include “webc.h”

int webc_DestroyCookie(**COOKIE_HANDLE** cookie);

name	- The name of the cookie
value	- Value the cookie will store
host	- Domain the cookie was received from
path	- Path of the cookie

Description:

This function removes a cookie that was instantiated with **webc_CreateCookie()**.

Returns:

0 if the cookie was successfully destroyed, otherwise -1

See Also:

webc_CreateCookie()

WEBC_DOCCLEAR

Function:

Clear all document content.

Summary:

#include “webc.h”

void webc_DocClear(**HDOC_HANDLE** doc, **HTML_BOOL** refresh);

doc	- The handle of the document to clear
refresh	- Boolean - whether to redraw the document window

Description:

Will discard the content of this document and free all associated memory.

Returns:

Nothing

See Also:

HDOC_HANDLE

WEBC_DOCCLOSE

Function:

Close an HTML Document after writing content.

Summary:

#include “webc.h”

void webc_DocClose(**HDOC_HANDLE** doc);

doc	- The handle of the document to be opened

Description:

This function should be called on an HTML Document as soon as the user is finnished writing HTML and text content to it (webc_DocWriteHtml(), webc_DocWriteString()).

Returns:

0 if the operation was successful, otherwise -1

See Also:

HDOC_HANDLE, webc_DocClear(), webc_DocOpen(), webc_DocWriteHtml(), webc_DocWriteString()

WEBC_DOCFINDELEMENT

Function:

Search for an element by id, name, type, and/or index.

Summary:

#include “webc.h”

HELEMENT_HANDLE webc_DocFindElement (**HDOC_HANDLE** doc,char* id, char* name, enum **HTMLElementType** type, long index);

doc	- The handle of the document to search
id	- The **ID** of the element to search for (NULL to match any id)
name	- The **NAME** of the element to search for (NULL to match any name)
type	- The **TYPE** of element to search for (**HTML_ELEMENT_ANY** to match any type)
index	- The index of the element to find (**HTML_INDEX_LAST** for the last element in the document)

Description:

Does a depth-first search of the document tree for an element matching all of the search criteria.

Returns:

The handle of the Nth element with the specified id, name, and/or type, where N is the above specified index; NULL if not found.

See Also:

HDOC_HANDLE, **HELEMENT_HANDLE**

WEBC_DOCGETBROWSER

Function:

Retrieve an HTML Document’s parent HTML Browser.

Summary:

#include “webc.h”

HBROWSER_HANDLE webc_DocGetBrowser (**HDOC_HANDLE** doc);

doc	- The handle of the document

Description:

Does a depth-first search of the document tree for an element matching all of the search criteria.

Returns:

The **HBROWSER_HANDLE** of the HTML Browser containing the given document.

See Also:

HDOC_HANDLE

WEBC_DOCOPEN

Function:

Open an HTML Document for writing.

Summary:

#include “webc.h”

int webc_DocOpen(**HDOC_HANDLE** doc);

doc	- The handle of the document to be opened

Description:

This function must preceed any calls to **webc_DocWriteHtml()** and **webc_DocWriteString()**.

Returns:

0 if the operation was successful, otherwise -1

See Also:

HDOC_HANDLE, webc_DocClear(), webc_DocClose(), webc_DocWriteHtml(), webc_DocWriteString()

WEBC_DOCREFRESH

Function:

Update the document and refresh the display.

Summary:

#include “webc.h”

void webc_DocRefresh(**HDOC_HANDLE** doc);

doc	- The handle of the document to refresh

Description:

Loads any outstanding URLs (including the source for the document) and redraws the document.

Returns:

Nothing

See Also:

HDOC_HANDLE

WEBC_DOCSETURL

Function:

Set the URL of a document.

Summary:

#include “webc.h”

void webc_DocSetUrl(**HDOC_HANDLE** doc, char* url, **HTML_BOOL** refresh);

doc	- The handle of the document to set the URL for.
url	- The URL of the document to load
refresh	- If TRUE, then load the given URL immediately and refresh the display; otherwise wait until **webc_DocRefresh()** is called.

Description:

Sets the URL of the specified document. The new page will not be loaded (unless ‘refresh’ is set to **HTML_TRUE**) until **webc_DocRefresh()** is called on this document.

Returns:

Nothing

See Also:

HDOC_HANDLE, webc_DocRefresh()

WEBC_DOCWRITEHTML

Function:

Write html code directly to a document.

Summary:

#include “webc.h”

long webc_DocWriteHtml(**HDOC_HANDLE** doc, char* html_src, long length, **HTML_BOOL** refresh);

doc	- The handle of the document
html_src	- Pointer to a null-terminated buffer of HTML source.
length	- The length of html_src.
refresh	- Whether to redraw the document window

Description:

This function will append HTML code onto the end of the existing document. The string passed to this function is treated exactly like HTML code arriving via HTTP or FILE.

Returns:

The number of characters successfully written to the document

See Also:

HDOC_HANDLE, webc_DocClear()

WEBC_DOCWRITESTRING

Function:

Write string directly to a document as is.

Summary:

#include "webc.h"

long webc_DocWriteString(**HDOC_HANDLE** doc, char* str, long length, **HTML_BOOL** refresh);

doc	- The handle of the document
str	- Pointer to a null-terminated string.
length	- The length of html_src.
refresh	- Whether to redraw the document window

Description:

This function will copy and change this string so that it will appear as is in the document.

Returns:

The number of characters successfully written to the document

See Also:

HDOC_HANDLE, webc_DocClear()

WEBC_ELEMENTDISABLE

Function:

Set an element to not be selectable.

Summary:

#include "webc.h"

void webc_ElementDisable(**HELEMENT_HANDLE** element, **HTML_BOOL** refresh);

element - The element to set
refresh - If true, then redraw this element

Description:

Will shade out an input widget and make it non-modifiable.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **VISIBILITY**, webc_ElementSetHidden()

WEBC_ELEMENTENABLE

Function:

Set an element to be selectable.

Summary:

#include "webc.h"

void webc_ElementEnable(**HELEMENT_HANDLE** element, **HTML_BOOL** refresh);

element	- The element to set
refresh	- If true, then redraw this element

Description:

Will shade in an input widget and make it modifiable.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **VISIBILITY**, webc_ElementSetHidden()

WEBC_ELEMENTGETBGCOLOR

Function:

Get an element's background color.

Summary:

#include "webc.h"

HTML_BOOL webc_ElementGetBgColor (**HELEMENT_HANDLE** element, struct html_color* color);

element	- The element whose background color to get
color	- Pointer to an html_color struct to fill in with the background color information for this element.

Description:

Get the element's background color.

Returns:

HTML_TRUE if color was filled with a valid color, HTML_FALSE otherwise

See Also:

HELEMENT_HANDLE, **COLOR, BGCOLOR**, webc_ElementSetColor()

WEBC_ELEMENTGETBGIMAGE

Function:

Get an element's background image url.

Summary:

#include "webc.h"

char* webc_ElementGetBgImage(**HELEMENT_HANDLE** element);

element	- The element whose background image url to get

Description:

This function only returns meaningful values for body, table, tr, and td elements.

Returns:

The element's background url or NULL if it has none

See Also:

HELEMENT_HANDLE, **BGIMAGE**, webc_ElementSetBgImage()

WEBC_ELEMENTGETCHECKED

Function:

Get the status of a checkbox or radio button.

Summary:

#include "webc.h"

char* webc_ElementGetChecked(**HELEMENT_HANDLE** element);

element	- The element whose value to get

Description:

If the given element handle is an **HTML_RADIO_BUTTON_ELEMENT** or a **HTML_CHECKBOX_ELEMENT**, then this function will return the element's value string (specified in the source HTML or through a call to ebc_ElementSetValue()) IF it is currently selected; if it is not currently selected, this function will return NULL.

Returns:

The element's value if checked or NULL if it is not checked

See Also:

HELEMENT_HANDLE, **VALUE**, **CHECKED**, webc_ElementSetChecked(), webc_ElementSetValue()

WEBC_ELEMENTGETCHILD

Function:

Search for a child of an element by id, name, type, and/or index.

Summary:

#include "webc.h"

HELEMENT_HANDLE webc_ElementGetChild (**HELEMENT_HANDLE** element, char* id, char* name, enum **HTMLElementType** type, long index);

element	- The parent element whose children to search
id	- The **ID** of the element to search for (NULL to match any id)
name	- The **NAME** of the element to search for (NULL to match any name)
type	- The **TYPE** of element to search for (**HTML_ELEMENT_ANY** to match any type)
index	- The index of the element to find (HTML_INDEX_LAST for the last element in the document)

Description:

Searches the document subtree at the given element for a child element that matches the given search criteria.

Returns:

The handle of the matching element or NULL if not found

See Also:

HELEMENT_HANDLE

WEBC_ELEMENTGETCOLOR

Function:

Get an element's foreground color.

Summary:

#include "webc.h"

HTML_BOOL webc_ElementGetColor(**HELEMENT_HANDLE** element, struct **html_color*** color);

element	- The element whose color to get
color	- Pointer to an html_color struct to fill in with the color information for this element.

Description:

Get the element's foreground color.

Returns:

HTML_TRUE if color was filled with a valid color, HTML_FALSE otherwise

See Also:

HELEMENT_HANDLE, **COLOR, BGCOLOR**, webc_ElementSetColor()

WEBC_ELEMENTGETDOCUMENT

Function:

Get the handle of the document containing a given element.

Summary:

#include "webc.h"

HDOC_HANDLE webc_ElementGetDocument (**HELEMENT_HANDLE** element);

element	- The element whose document handle to return

Description:

Gets the handle of the document that contains the given element.

Returns:

Handle of the parent document of this element.

See Also:

HELEMENT_HANDLE, **HDOC_HANDLE**

WEBC_ELEMENTGETFIRSTCHILD

Function:

Get the first child of an element.

Summary:

#include "webc.h"

HELEMENT_HANDLE webc_ElementGetFirstChild (**HELEMENT_HANDLE** element);

element	- The parent element

Description:

Returns the first child element of the given element.

Returns:

The handle of the first child element or NULL if no children

See Also:

HELEMENT_HANDLE, webc_ElementGetParent(), webc_ElementGetLastChild(), webc_ElementGetNextSibling(), webc_ElementGetPrevSibling()

WEBC_ELEMENTGETFRAMEDOCUMENT

Function:

Get a frame's child document.

Summary:

#include "webc.h"

HDOC_HANDLE webc_ElementGetFrameDocument (**HELEMENT_HANDLE** element);

element	- The frame element handle

Description:

Gets a frame's child document.

Returns:

The handle of the HTML Document or NULL if this element is not a frame

See Also:

HELEMENT_HANDLE, **HDOC_HANDLE**, webc_BrowserGetDocument()

WEBC_ELEMENTGETHEIGHT

Function:

Get an element's height in pixels.

Summary:

#include "webc.h"

long webc_ElementGetHeight(**HELEMENT_HANDLE** element);

element	- The element whose height to get

Description:

Get an element's height in pixels.

Returns:

The element's height in pixels

See Also:

HELEMENT_HANDLE, **HEIGHT, WIDTH**, webc_ElementSetHeight(), webc_ElementGetWidth(), webc_ElementSetWidth()

WEBC_ELEMENTGETID

Function:

Get an element’s unique id.

Summary:

#include “webc.h”

char* webc_ElementGetId(**HELEMENT_HANDLE** element);

element	- The element whose ID to get

Description:

Returns the unique id string of this element.

Returns:

The element’s id or NULL if it doesn’t have one

See Also:

HELEMENT_HANDLE, **ID**

WEBC_ELEMENTGETLASTCHILD

Function:

Get the last child of an element.

Summary:

#include “webc.h”

HELEMENT_HANDLE webc_ElementGetLastChild (**HELEMENT_HANDLE** element);

element	- The parent element

Description:

Returns the last child element of the given element.

Returns:

The handle of the last child element or NULL if no children

See Also:

HELEMENT_HANDLE, webc_ElementGetParent(), webc_ElementGetFirstChild(), webc_ElementGetNextSibling(), webc_ElementGetPrevSibling()

WEBC_ELEMENTGETNAME

Function:

Get an element's name.

Summary:

#include "webc.h"

char* webc_ElementGetName(**HELEMENT_HANDLE** element);

element	- The element whose name to get

Description:

Returns the name string of this element.

Returns:

The element's name or NULL if it doesn't have one

See Also:

HELEMENT_HANDLE, **NAME**, webc_ElementSetName()

WEBC_ELEMENTGETNEXTSIBLING

Function:

Get the next sibling of an element.

Summary:

#include "webc.h"

HELEMENT_HANDLE webc_ElementGetNextSibling (**HELEMENT_HANDLE** element);

element	- The element whose sibling to get

Description:

Returns the next sibling element of the given element.

Returns:

The handle of the next sibling element or NULL if this is the last child

See Also:

HELEMENT_HANDLE, webc_ElementGetParent(), webc_ElementGetFirstChild(), webc_ElementGetLastChild(), webc_ElementGetPrevSibling()

WEBC_ELEMENTGETPARENT

Function:

Get the immediate parent of an element.

Summary:

#include "webc.h"

HELEMENT_HANDLE webc_ElementGetParent (HELEMENT_HANDLE element);

element - The element whose parent to get

Description:

Returns the direct parent of an element in the document tree.

Returns:

The handle of the parent element or NULL if none

See Also:

HELEMENT_HANDLE, webc_ElementGetFirstChild(), webc_ElementGetLastChild(), webc_ElementGetNextSibling(), webc_ElementGetPrevSibling()

WEBC_ELEMENTGETPOSITION

Function:

Get an element's relative position.

Summary:

#include "webc.h"

void webc_ElementGetPosition (**HELEMENT_HANDLE** element, long* px, long* py);

element - The element whose position to get

px - Pointer to a long to set to the current x position

py - Pointer to a long to set to the current y position

Description:

Gets an element's position in pixels relative to its parent element.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **POSITION**, webc_ElementSetPosition()

WEBC_ELEMENTGETPREVSIBLING

Function:

Get the prev sibling of an element.

Summary:

#include “webc.h”

HELEMENT_HANDLE webc_ElementGetPrevSibling (**HELEMENT_HANDLE** element);

element	- The element whose sibling to get

Description:

Returns the prev sibling element of the given element.

Returns:

The handle of the prev sibling element or NULL if this is the first child

See Also:

HELEMENT_HANDLE, webc_ElementGetParent(), webc_ElementGetFirstChild(), webc_ElementGetLastChild(), webc_ElementGetNextSibling()

WEBC_ELEMENTGETSRC

Function:

Get the source url for an image or frame element.

Summary:

#include “webc.h”

char* webc_ElementGetSrc(**HELEMENT_HANDLE** element);

element - The element whose src to get

Description:

This function only returns meaningful values for frame and image elements.

Returns:

The element’s src or NULL if it has none

See Also:

HELEMENT_HANDLE, **SRC**, webc_ElementSetSrc()

WEBC_ELEMENTGETTYPE

Function:

Get element type.

Summary:

#include "webc.h"

char* webc_ElementGetType(**HELEMENT_HANDLE** element);

element - The element whose type to return

Description:

Returns element's type.

Returns:

The element type

See Also:

HELEMENT_HANDLE, HTMLElementType()

WEBC_ELEMENTGETVALUE

Function:

Get the value of an input widget.

Summary:

#include "webc.h"

char* webc_ElementGetValue**(HELEMENT_HANDLE** element);

element - The element whose value to get

Description:

If the given element handle is of type HTML_EDIT_STR_ELEMENT (text field), HTML_EDITBOX_ELEMENT (text box), or HTML_SELECT_ELEMENT (combo box), then this function will return the currently entered value of the widget. For element types that have no input value (everything that is not a widget), this function will return NULL. To retrieve the status of boolean type input widgets (such as radio buttons and checkboxes), use **webc_ElementGetChecked()**.

Returns:

The element's value or NULL if it doesn't have one

See Also:

HELEMENT_HANDLE, **VALUE**, webc_ElementSetValue(), webc_ElementGetChecked()

WEBC_ELEMENTGETWIDTH

Function:

Get an element's width in pixels.

Summary:

#include "webc.h"

long webc_ElementGetWidthlong webc_ElementGetWidth (**HELEMENT_HANDLE** element);

element	- The element whose width to get

Description:

Gets an element's width in pixels.

Returns:

The element's width in pixels

See Also:

HELEMENT_HANDLE, **WIDTH, HEIGHT**, webc_ElementSetWidth(), webc_ElementGetHeight(), webc_ElementSetHeight()

WEBC_ELEMENTNEXT

Function:

Find next element in document.

Summary:

#include "webc.h"

HELEMENT_HANDLE webc_ElementNext (**HELEMENT_HANDLE** element, char* id, char* name, enum **HTMLElementType** type, long index);

element	- The element to search from
id	- The **ID** of the element to search for (NULL to match any id)
name	- The **NAME** of the element to search for (NULL to match any name)
type	- The **TYPE** of element to search for (**HTML_ELEMENT_ANY** to match any type)
index	- The offset from the given element of the element to find (0 to match the first element after this one matching all the other criteria)

Description:

Searches the document this element is a part of, starting at the given element, for the next element that matches all the given criteria.

Returns:

The handle of the matching element or NULL if not found

See Also:

HELEMENT_HANDLE, webc_ElementPrev(), webc_ElementGetHeight(), webc_ElementSetHeight()

WEBC_ELEMENTPREV

Function:

Find prev element in document.

Summary:

#include “webc.h”

HELEMENT_HANDLE webc_ElementPrev (**HELEMENT_HANDLE** element, char* id, char* name, enum **HTMLElementType** type, long index);

element	- The element to search from
id	- The **ID** of the element to search for (NULL to match any id)
name	- The **NAME** of the element to search for (NULL to match any name)
type	- The **TYPE** of element to search for (**HTML_ELEMENT_ANY** to match any type)
index	- The offset from the given element of the element to find (0 to match the last element before this one matching all the other criteria)

Description:

Searches the document this element is a part of, starting at the given element, for the previous element that matches all the given criteria.

Returns:

The handle of the matching element or NULL if not found

See Also:

HELEMENT_HANDLE, webc_ElementNext()

WEBC_ELEMENTREFRESH

Function:

Update/Redraw an element.

Summary:

#include “webc.h”

void webc_ElementRefresh(**HELEMENT_HANDLE** element);

element	- The element to refresh

Description:

Loads any outstanding URLs associated with this element (such as image/frame source or background image) and redraws it and its children on the screen.

Returns:

Nothing

See Also:

HELEMENT_HANDLE

WEBC_ELEMENTSETBGCOLOR

Function:

Set an element's background color.

Summary:

#include "webc.h"

void webc_ElementSetBgColor(**HELEMENT_HANDLE** element, struct **html_color*** color, **HTML_BOOL** refresh);

element	- The element to set
color	- Pointer to html_color struct with new background color
refresh	- If true, then redraw this element

Description:

Sets the background color for the given element and its children.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **BGCOLOR**, webc_ElementGetBgColor(), webc_ElementRefresh()

WEBC_ELEMENTSETBGIMAGE

Function:

Set an element's background image url.

Summary:

#include "webc.h"

void webc_ElementSetBgImage(**HELEMENT_HANDLE** element, char* bgimg, **HTML_BOOL** refresh);

element	- The element to set
bgimg	- The new background image url
refresh	- If true, then load the new url and redraw this element

Description:

Sets the background image url for the specified element. Only affects HTML_BODY_ELEMENT, HTML_TABLE_ELEMENT, HTML_TABLE_CELL_ELEMENT, and HTML_TABLE_ROW_ELEMENT. Doesn't load the new bitmap or redraw (unless 'refresh' is set to true) until **webc_ElementRefresh()** is called.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **BGIMAGE**, webc_ElementGetBgImage(), webc_ElementRefresh()

WEBC_ELEMENTSETCHECKED

Function:

Set checked status for checkbox or radio button.

Summary:

#include "webc.h"

void webc_ElementSetChecked(**HELEMENT_HANDLE** element, **HTML_BOOL** checked, **HTML_BOOL** refresh);

element	- The element to set
checked	- Whether or not the element should be checked
refresh	- If true, then redraw this element

Description:

If the element handle passed to this routine is that of an **HTML_CHECKBOX_ELEMENT**, or an HTML_RADIO_BUTTON_ELEMENT, then the 'checked' status of the element will be set according to the boolean 'checked' parameter. If this function is called with a 'checked' value of HTML_TRUE, then subsequent calls to **webc_ElementGetChecked()** will return the string set as the value for this element, via **webc_ElementSetValue()**. If HTML_FALSE is passed to this routine, then subsequent calls to **webc_ElementGetChecked()** will return NULL.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **CHECKED**, webc_ElementGetChecked(), webc_ElementSetValue(), webc_ElementRefresh()

WEBC_ELEMENTSETCOLOR

Function:

Set an element's foreground color.

Summary:

#include "webc.h"

void webc_ElementSetColor(**HELEMENT_HANDLE** element, struct **html_color*** color, **HTML_BOOL** refresh);

element	- The element to set
color	- Pointer to **html_color** struct with new foreground color
refresh	- If true, then redraw this element

Description:

Sets the foreground color for the given element and its children.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **COLOR**, webc_ElementGetColor(), webc_ElementRefresh()

WEBC_ELEMENTSETFOCUS

Function:

Set the current input focus to an element.

Summary:

#include “webc.h”

void webc_ElementSetFocus(**HELEMENT_HANDLE** element,**HTML_BOOL** refresh);

element	- The element to set
refresh	- If true, then redraw this element

Description:

Makes the given element the input focus for keyboard and mouse events.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, webc_ElementRefresh()

WEBC_ELEMENTSETHEIGHT

Function:

Set an element’s height (Deprecated).

Summary:

#include “webc.h”

void webc_ElementSetHeight(**HELEMENT_HANDLE** element, long h, **HTML_BOOL** refresh);

element	- The element to set
w	- Height in pixels
refresh	- If true, then redraw this element

Description:

Sets an element’s height in pixels.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **HEIGHT**, webc_ElementGetHeight(), webc_ElementGetWidth(), webc_ElementSetWidth()

WEBC_ELEMENTSETHIDDEN

Function:

Set an element to not display.

Summary:

#include “webc.h”

void webc_ElementSetHidden(**HELEMENT_HANDLE** element, **HTML_BOOL** refresh);

element	- The element to set
refresh	- If true, then redraw this element

Description:

Effectively hides an element and its effects on surrounding elements (e.g. <BR>).

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **VISIBILITY**, webc_ElementSetVisible()

WEBC_ELEMENTSETINLINE

Function:

Place an element in the flow of text.

Summary:

#include “webc.h”

void webc_ElementSetInline(**HELEMENT_HANDLE** element, **HTML_BOOL** refresh);

element	- The element to set
refresh	- If true, then redraw this element

Description:

When **webc_ElementSetPosition()** is called, an element is taken out of the text flow so that its position will stay as the user has specified when the page is reformatted and redrawn. This function negates the affects of **webc_ElementSetPosition()** by placing the given element back into the flow of text.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, webc_ElementSetPosition(), webc_ElementRefresh()

WEBC_ELEMENTSETNAME

Function:

Set an element's name.

Summary:

#include "webc.h"

void webc_ElementSetName(**HELEMENT_HANDLE** element, char* name, **HTML_BOOL** refresh);

element	- The element to set
name	- The new name
refresh	- If true, then redraw this element

Description:

Sets the name attribute for the specified element.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **NAME**, webc_ElementGetName(), webc_ElementRefresh()

WEBC_ELEMENTSETPOSITION

Function:

Set an element's position in pixels.

Summary:

#include "webc.h"

void webc_ElementSetPosition(**HELEMENT_HANDLE** element, long x, long y, **HTML_BOOL** refresh);

element	- The element whose position to set
x	- Relative X position in pixels
y	- Relative Y position in pixels
refresh	- If true, then redraw this element

Description:

Sets an element's position in pixels relative to its parent element. Also removes the given element from the flow of text in the document so that the specified position will be retained when the document is reformatted and redrawn (use **webc_ElementSetInline()** to put the element back into the flow of text).

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **POSITION**, webc_ElementGetPosition(), webc_ElementSetInline()

WEBC_ELEMENTSETSELECTED

(**HELEMENT_HANDLE** element, int selected, **HTML_BOOL** refresh);

Function:

Set the selected option in a combo box.

Summary:

#include “webc.h”

void webc_ElementSetSelected(**HELEMENT_HANDLE** element, int selected, **HTML_BOOL** refresh);

element	-	The element to set.
selected	-	Index of the option to select.
refresh	-	If true, then redraw this element

Description:

Sets the index (beginning with 0) of the selected item in a combo box. The value (string) of the selected item of a combo box can be retrieved via webc_ElementGetValue().

Returns:

Nothing

See Also:

HELEMENT_HANDLE, SELECTED, webc_ElementGetValue()

WEBC_ELEMENTSETSRC

Function:

Set the source url for an image or frame element.

Summary:

#include “webc.h”

void webc_ElementSetSrc(**HELEMENT_HANDLE** element, char* src, **HTML_BOOL** refresh);

element	-	The element to set
src	-	The new src
refresh	-	If true, then load the new url and redraw this element

Description:

Sets src attribute for the specified element. Affects HTML_FRAME_ELEMENT and HTML_IMAGE_ELEMENT. Doesn’t load or redraw (unless ‘refresh’ is set to true) until **webc_ElementRefresh()** is called.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **SRC**, webc_ElementGetSrc(), webc_ElementRefresh()

WEBC_ELEMENTSETSTYLE

Function:

Set style properties for an element.

Summary:

#include "webc.h"

void webc_ElementSetStyle(**HELEMENT_HANDLE** element, char* style, **HTML_BOOL** refresh);

element	- The element to set style info for
style	- Cascading style sheet source data (e.g. "a.class1 { font-size: 12pt; }")
refresh	- If true, then redraw this element

Description:

This function will be operational once CSS support has been added to Webster.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **SRC**, **STYLE**

WEBC_ELEMENTSETVALUE

Function:

Set the current value for an input widget.

Summary:

#include "webc.h"

void webc_ElementSetValue(**HELEMENT_HANDLE** element, char* value, **HTML_BOOL** refresh);

element	- The element whose value to get
value	- The string to set the element's value to
refresh	- If true, then redraw this element

Description:

For text entry widgets (HTML_EDIT_STR_ELEMENT, HTML_EDITBOX_ELEMENT), this function sets the current text in the widget. For boolean-type widgets (HTML_CHECKBOX_ELEMENT, HTML_RADIO_BUTTON_ELEMENT), this function will set the value string that will be returned by **webc_ElementGetChecked()** if the element's status is 'checked.'

Use **webc_ElementSetSelected()** to set the currently selected item in combo boxes (HTML_SELECT_ELEMENT), use **webc_ElementSetSelected()**.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **VALUE**, webc_ElementGetValue(), webc_ElementGetChecked(), webc_ElementSetChecked(), webc_ElementSetSelected(), webc_ElementRefresh()

WEBC_ELEMENTSETVISIBLE

Function:

Set an element to display.

Summary:

#include “webc.h”

void webc_ElementSetVisible (**HELEMENT_HANDLE** element, **HTML_BOOL** refresh);

element	- The element to set
refresh	- If true, then redraw this element

Description:

Undoes the affects of **webc_ElementSetHidden()**, so that an element and its effects on surrounding elements will again be shown.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **VISIBILITY**, webc_ElementSetHidden()

WEBC_ELEMENTSETWIDTH

Function:

Set an element’s width (Deprecated).

Summary:

#include “webc.h”

void webc_ElementSetWidth(**HELEMENT_HANDLE** element, long w, **HTML_BOOL** refresh);

element	- The element to set
w	- Width in pixels
refresh	- If true, then redraw this element

Description:

Sets an element’s width in pixels.

Returns:

Nothing

See Also:

HELEMENT_HANDLE, **WIDTH**, webc_ElementGetWidth(), webc_ElementSetHeight(), webc_ElementGetHeight()

WEBC_EXIT(VOID);

Function:

Free resources used by the Web Client.

Summary:

#include "webc.h"

int webc_Exit(void);

no parameters

Description:

This function should be called when then application exits.

Returns:

0 if successful

See Also:

webc_Init()

WEBC_FINDCOOKIE

Function:

Find a cookie that has the specified attributes.

Summary:

#include "webc.h"

COOKIE_HANDLE webc_FindCookie(char* name, char* value, char* host, char* path, int num);

name	- The name of the cookie
value	- Value the cookie stores
host	- Domain the cookie was received from
path	- Path of the cookie
num	- Sequence number of the cookie (beginning from 0)

Description:

Find a cookie that has the specified attributes. If more than one cookie matches, then the NUMth cookie will be returned.

Returns:

0 if the cookie was not found, or a handle to the cookie otherwise

See Also:

WEBC_INIT(VOID);

Function:

Initialize the Web Client.

Summary:

#include "webc.h"

int webc_Init(void);

no parameters

Description:

This function must be called before any other API calls are made.

Returns:

0 if successful, -1 otherwise

See Also:

webc_Exit()

WEBC_JSCRIPTDEFINEFUNCTION

Function:

Call a native C function from Javascript.

Summary:

#include "webc.h"

void webc_JScriptDefineFunction(**HDOC_HANDLE** doc, const char* name, JSNative func, int argc);

doc	- Any created document
name	- The name that the javascript interpretor will recognize.
func	- The native function that will be called.
argc	- Parameter count

Description:

This function allows a native C function to be called by javascript. This function must have this signature: JSBool someName(JSContext *cx, JSObject *obj, uintN argc, jsval *argv, jsval *rval); When the function executes, cx will be the JSContext that is associated with the executing script. obj is the global object that can access all other objects in running scripts. argc is the number of arguments passed into the function. argv is an array of jsvalues that contain the parameters passed into the function. rval should be set by the user and will be the return value. The function should return JS_TRUE.

Returns:

Nothing

See Also:

WEBC_JSCRIPTGETDOC

Function:

Get the HTMLDocument handle from a JSFunction.

Summary:

#include "webc.h"

HDOC_HANDLE webc_JScriptGetDoc(struct JSContext* cx);

cx	- The javascript context that was running

Description:

This function should be used to get the **HDOC_HANDLE** associated with the javascript context that was running. Use this handle to access the rest of the API's functionality.

Returns:

Returns: a handle to the document

See Also:

WEBC_REGISTEREVENTCALLBACK

Function:

Set a function to use as a custom event handler.

Summary:

#include "webc.h"

int webc_RegisterEventCallback(HtmlEventCallback cb, RTP_PFCHAR name);

cb	- The event handler callback.
name	- The name of this event handler

Description:

After this function is called, whenever an HTML tag loaded into Webster specifies its 'eventhandler' attribute (a Webster HTML extension) as the name passed in, all events that happen to that tag will be passed on to the C function specified as cb.

Returns:

0 if the callback was successfully registered; otherwise -1

See Also:

HELEMENT_HANDLE, SELECTED, webc_ElementGetValue()