INSIDE MACINTOSH

HTML Rendering Library Preliminary Documentation

For Mac OS 9

Preliminary Documentation

October 26, 1999
Technical Publications

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

HTML Rendering Library Reference 5

HTML Rendering Functions 5
- Identifying The HTML Rendering Library 5
  - HRHTMLRenderingLibAvailable 6
  - HRGetHTMLRenderingLibVersion 6
- Obtaining and Disposing of HRReference Values 6
  - HRNewReference 6
  - HRDisposeReference 7
- Formatting the Rendering Area 7
  - HRSetDrawBorder 8
  - HRSetGrowboxCutout 8
  - HRSetScrollbarState 9
- Setting and Updating the Rendering Area 10
  - HRSetRenderingRect 10
  - HRGetRenderedImageSize 11
  - HRDraw 12
  - HRScrollToLocation 12
- Setting Graphics and Display Options 13
  - HRScreenConfigurationChanged 13
  - HRSetGrafPtr 13
  - HRForceQuickdraw 14
  - HRFreeMemory 15
- Working With Events 15
  - HRIsHREvent 15
  - HRActivate 16
  - HRDeactivate 16
- Navigating HTML Pages 17
  - HRGoToFile 17
  - HRGoToURL 18
  - HRGoToPtr 19
  - HRGoToAnchor 20
- Obtaining Information About Pages 20
  - HRGetHTMLURL 21

HRGetRootURL 21
HRGetBaseURL 22
HRGetTitle 23
HRGetHTMLFile 23
Converting URL and FSSpec Data 24
HRUtilCreateFullURL 24
HRUtilGetFSSpecFromURL 25
HRUtilGetURLFromFSSpec 26
Application-Defined Functions 26
Handling Newly Visited Links 27
MyNewURLProc 27
NewHRNewURLUPP 28
HRRegisterNewURLUPP 28
HRUnregisterNewURLUPP 29
DisposeHRNewURLUPP 29
Handling Previously Visited Links 30
MyHRWasURLVisitedProc 30
NewHRWasURLVisitedUPP 31
HRRegisterWasURLVisitedUPP 31
HRUnregisterWasURLVisitedUPP 32
DisposeHRWasURLVisitedUPP 32
Intercepting and Redirecting URL's 32
MyHRURLToFSSpecProc 32
NewHRURLToFSSpecUPP 34
HRRegisterURLToFSSpecUPP 35
HRUnregisterURLToFSSpecUPP 35
DisposeHRURLToFSSpecUPP 35
HTML Rendering Library Constants 36
Scrollbar State Constants 36
URL Source Type Constants 36
Result Codes 37

HTML Rendering Library Reference

The HMTL Rendering Library application programming interface provides basic rendering functionality; that is, it gives your application the ability to draw text and images in a window, as specified by HTML data. Note that this functionality does not include other features of HTML browsers, such as history tracking or plug-in support.

HTML Rendering Functions

The kinds of tasks you can perform with the HTML Rendering Library API include:

Identifying The HTML Rendering Library
Obtaining and Disposing of HRReference Values
Formatting the Rendering Area
Setting and Updating the Rendering Area
Setting Graphics and Display Options
Working With Events
Navigating HTML Pages
Obtaining Information About Pages
Converting URL and FSSpec Data

Identifying The HTML Rendering Library

The HTML Rendering Library provides the following functions to report its availability:

- `HRHTMLRenderingLibAvailable`
- `HRGetHTMLRenderingLibVersion`

HRHTMLRenderingLibAvailable

Reports whether the HTML Rendering Library is available.

```
pascal Boolean HRHTMLRenderingLibAvailable()
```

function result

Returns `true` if the HTML Rendering Library is available; returns `false` otherwise.

HRGetHTMLRenderingLibVersion

Identifies which version of the HTML Rendering Library is available.

```
pascal OSStatus HRGetHTMLRenderingLibVersion(NumVersion *returnVers)
```

`returnVers`

A pointer to a `NumVersion` structure. On return, this structure contains a value identifying the installed version of the HTML Rendering Library.

function result

A result code. For a list of possible return values, see Result Codes.

Obtaining and Disposing of HRReference Values

The HTML Rendering Library provides the following functions to obtain and dispose of HRReference values:

- `HRNewReference`
- `HRDisposeReference`

HRNewReference

Obtains a new `HRReference`.

```
pascal OSStatus HRNewReference(
                    HRReference *hrRef,
                    OSType      rendererType,
                    grafPtr     GrafPtr)
```

`hrRef`

A pointer to a `HRReference` value. On return, your application uses this value to call other functions in the HTML Rendering Library.

`rendererType`

A value of type `OSType` identifying the type of HTML renderer available. Currently, the constant `kHRRendererHTML32Type` is the only value you may pass in this parameter.

`grafPtr`

A pointer to a valid `GrafPort`.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

When your application has no further need for a particular `HRReference`, dispose of it by calling the function `HRDisposeReference`.

HRDisposeReference

Disposes of a previously obtained `HRReference`.

```
pascal OSStatus HRDisposeReference(HRReference hrRef)
```

`hrRef`

A `HRReference` value previously obtained by your application.

function result

A result code. For a list of possible return values, see Result Codes.

Formatting the Rendering Area

The HTML Rendering Library provides the following functions to format the rendering area:

- HRSetDrawBorder
- HRSetGrowboxCutout
- HRSetScrollbarState

HRSetDrawBorder

Specifies whether to draw a border around the rendering area.

```
pascal OSStatus HRSetDrawBorder(
                    HRReference hrRef,
                    Boolean drawBorder);
```

`hrRef`

A `HRReference` value previously obtained by your application.

`drawBorder`

If you pass `true`, the HTML Rendering Library draws a border around the rendering area. If you pass `false`, the HTML Rendering Library draws the rendering area without a border. The default setting is `false`.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

You may use this optional function to specify whether a border is drawn around the rendering area. This might be useful when you specify a rendering area in a window containing other elements, for example. The default setting is to render without a border.

HRSetGrowboxCutout

Specifies whether to allow for a size box when drawing scrollbars.

```
pascal OSStatus HRSetGrowboxCutout(
                    HRReference  hrRef,
                    Boolean      allowCutout);
```

`hrRef`

A `HRReference` value previously obtained by your application.

`allowCutout`

If you pass `true` in this parameter, the HTML Rendering Library draws scroll bars that provide space for a size box. If you pass `false` in this parameter, the HTML Rendering Library draws scroll bars that extend all the way to the bottom right corner of the rendering area. The default setting is `false`.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

You may find it useful to specify a cutout for a size box if the HTML rendering area you specify extends to the lower right corner of a window. The default setting is to draw scrollbars without leaving room for a size box.

HRSetScrollbarState

Specifies how scrollbars are drawn.

```
pascal OSStatus HRSetScrollbarState(
                    HRReference          hrRef,
                    HRScrollbarState     hScrollbarState,
                    HRScrollbarState     vScrollbarState);
```

`hrRef`

A `HRReference` value previously obtained by your application.

`hScrollbarState`

A value of type `HRScrollbarState`. See Discussion for a list of the constants you may pass in this parameter.

`vScrollbarState`

A value of type `HRScrollbarState`. See Discussion for a list of the constants you may pass in this parameter.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

You can specify one of three values for scrollbars:

- `eHRScrollbarOn` tells the HTML Rendering Library to draw scrollbars at all times. If the data does not fill the current view, the scroll bars are inactive.
- `eHRScrollbarOff` tells the HTML Rendering Library never to draw scrollbars. You may find this option useful if if the HTML rendering area you specify does not extend to the edge of a window.
- `eHRScrollbarAuto` tells the HTML Rendering Library to draw scrollbars as needed. This is the default setting.

Setting and Updating the Rendering Area

The HTML Rendering Library provides the following functions to set and update the rendering area:

- `HRSetRenderingRect`
- `HRGetRenderedImageSize`
- `HRDraw`
- `HRScrollToLocation`

HRSetRenderingRect

Specifies the boundaries of the HTML rendering area.

```
pascal OSStatus HRSetRenderingRect(
                    HRReference  hrRef,
                    const Rect   *renderingRect)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`renderingRect`

A pointer to a value of type `Rect`. This value specifies the boundaries of the HTML rendering area. Specify the boundaries in the `GrafPort's` port coordinates.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

The HTML Rendering Library draws all elements, including scroll bars, inside the area you specify with the `HRSetRenderingRect` function.

HRGetRenderedImageSize

Reports the size of the rendered image.

```
pascal OSStatus HRGetRenderedImageSize(
                    HRReference  hrRef,
                    Point        *renderingSize)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`renderingSize`

A pointer to a value of type `Point`. On return, the HTML Rendering Library uses this value to report the size of the entire HTML page; that is, the smallest rectangle that fully encloses the entire rendered page (as drawn without scrollbars).

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

The HTML Rendering Library attempts to limit the width of the rendered page to the width of the rectangle you specify by calling the `HRSetRenderingRect` function. The length of the page is calculated based on the resulting width.

HRDraw

Updates a specified region for a given `HRReference`.

```
pascal OSStatus HRDraw(
                    HRReference hrRef,
                    RgnHandle   updateRgnH)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`updateRgnH`

A handle of type `RgnHandle`. This handle specifies the region that you want the HTML Rendering Library to update. Specify the region in the `GrafPort's` local coordinates. If you pass `NULL`, the HTML Rendering Library will update the entire rectangle you previously specified with the `HRSetRenderingRect` function.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

Be sure to specify the dimensions of the rendering area by calling the `HRSetRenderingRect` function at least once before you call the `HRDraw` function.

HRScrollToLocation

Scrolls to a specified view of the HTML rendering area.

```
pascal OSStatus HRSetRenderingRect(
                    HRReference  hrRef,
                    Point        *location)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`location`

A pointer to a value of type `Point`. Your application uses this value to specify the upper left corner of the visible portion of the rendering area.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

The HTML Rendering Library limits scrolling to the available rendering area.

Setting Graphics and Display Options

The HTML Rendering Library provides the following functions to set graphics and display options:

- `HRScreenConfigurationChanged`
- `HRSetGrafPtr`
- `HRForceQuickdraw`
- `HRFreeMemory`

HRScreenConfigurationChanged

Informs the HTML Rendering Library that the screen depth has changed.

```
pascal void HRScreenConfigurationChanged()
```

DISCUSSION

Call the `HRScreenConfigurationChanged` function every time the screen depth changes or the user's monitor configuration changes. This allows the HTML Rendering Library to redraw correctly.

HRSetGrafPtr

Sets a new `GrafPort` for a given `HRReference`.

```
pascal OSStatus HRSetGrafPtr(
                    HRReference hrRef,
                    grafPtr     GrafPtr)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`grafPtr`

A pointer to a valid `GrafPort`. This value replaces the `GrafPort` previously specified for the given `HRReference`.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

Call this function whenever the `GrafPort` changes for a given `HRReference`, as occurs during printing, for example. Be sure to call the `HRSetGrafPtr` function again to reset the original `GrafPort` when necessary.

HRForceQuickdraw

Forces all images to be drawn with QuickDraw.

```
pascal OSStatus HRForceQuickdraw(
                    HRReference  hrRef,
                    Boolean      forceQuickdraw)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`forceQuickdraw`

Pass a value of `true` in this parameter to force all images to be drawn with QuickDraw.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

This function may be useful when working with print drivers that require QuickDraw for image capturing.

HRFreeMemory

Attempts to release cache memory for use by your application.

```
pascal SInt32 HRFreeMemory(Size inBytesNeeded)
```

`inBytesNeeded`

A value indicating how many bytes of cache memory your application desires to obtain.

function result

A value indicating how many bytes of cache memory the HTML Rendering Library makes available to your application.

DISCUSSION

Call the `HRFreeMemory` function to flush images and movies from the HTML Rendering Library cache. Be sure to check the function result to determine whether sufficient memory was made available to your application.

Working With Events

The HTML Rendering Library provides the following functions to work with events:

- `HRIsHREvent`
- `HRActivate`
- `HRDeactivate`

HRIsHREvent

Gives the HTML Rendering Library an opportunity to handle events.

```
pascal Boolean HRIsHREvent(const EventRecord *eventRecord)
```

`eventRecord`

A pointer to an `EventRecord`.

function result

Returns `true` if the HTML Rendering Library handles the specified event; returns `false` otherwise.

DISCUSSION

Call the `HRIsHREvent` function with every event received by your application. This ensures that the HTML Rendering Library has an opportunity to handle user clicks and cursor changes.

HRActivate

Activates the window associated with a given `HRReference`.

```
pascal OSStatus HRActivate(HRReference hrRef)
```

`hrRef`

A `HRReference` value previously obtained by your application.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

Call this function whenever the window associated with a given `HRReference` becomes active. This allows the HTML Rendering Library to activate scrollbars and handle events as appropriate.

HRDeactivate

Deactivates the window associated with a given `HRReference`.

```
pascal OSStatus HRDeactivate(HRReference hrRef)
```

`hrRef`

A `HRReference` value previously obtained by your application.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

Call this function whenever the window associated with a given `HRReference` becomes inactive.

Navigating HTML Pages

The HTML Rendering Library provides the following functions to navigate HTML pages:

- `HRGoToFile`
- `HRGoToURL`
- `HRGoToPtr`
- `HRGoToAnchor`

HRGoToFile

Specifies an HTML file for rendering.

```
pascal OSStatus HRGoToFile(
                    HRReference  hrRef,
                    const FSSpec *fsspec,
                    Boolean      addToHistory,
                    Boolean      forceRefresh)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`fsspec`

A pointer to a file system specification record (`FSSpec`) for the HTML file that you wish to have rendered by the HTML Rendering Library.

`addToHistory`

Pass `true` in this parameter if you wish to have this file added to the list of visited links.

`forceRefresh`

Pass `true` in this parameter if you wish to force a screen update.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

Note that the HTML Rendering Library calls your `MyNewURLProc` application-defined function (if you use one) with this `FSSpec`. If you need to pass a URL instead of an `FSSpec`, use the `HRGoToURL` function.

HRGoToURL

Renders a local file specified as a URL.

```
pascal OSStatus HRGoToURL(
                    HRReference  hrRef,
                    const char   *url,
                    Boolean      addToHistory,
                    Boolean      forceRefresh)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`url`

A pointer to a C string containing a Universal Resource Locator (URL).

`addToHistory`

Pass `true` in this parameter if you wish to have this file added to the list of visited links.

`forceRefresh`

Pass `true` in this parameter if you wish to force a screen update.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

Use this function for rendering local files. The URL should begin with 'file:///'. Note that the HTML Rendering Library calls your `MyNewURLProc` application-defined function (if you use one) with this URL. If you need to pass an `FSSpec` instead of a URL, use the `HRGoToFile` function.

HRGoToPtr

Renders a set of HTML data from a specified buffer.

```
pascal OSStatus HRGoToPtr(
                    HRReference  hrRef,
                    char         *buffer,
                    UInt32       bufferSize,
                    Boolean      addToHistory,
                    Boolean      forceRefresh)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`buffer`

A pointer to a buffer containing HTML data.

`bufferSize`

A value specifying the number of bytes in the buffer.

`addToHistory`

Pass `true` in this parameter if you wish to have this data added to the list of visited links.

`forceRefresh`

Pass `true` in this parameter if you wish to force a screen update.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

The HTML Rendering Library makes a copy of the HTML source data for its use. Your application is responsible for disposing of the original buffer. Note that any relative links you specify in the HTML data are relative to your application, as the HTML data has no associated URL.

HRGoToAnchor

Specifies an HTML anchor to scroll into view.

```
pascal OSStatus HRGoToFile(
                    HRReference  hrRef,
                    const char   *anchorName)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`anchorName`

A pointer to a C string containing the name of an anchor on the current HTML page.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

Note that anchors are case sensitive.

Obtaining Information About Pages

The HTML Rendering Library provides the following functions to obtain information about HTML pages:

- `HRGetHTMLURL`
- `HRGetRootURL`

- HRGetBaseURL
- HRGetTitle
- HRGetHTMLFile

HRGetHTMLURL

Obtains the URL for a given HTML page.

```
pascal OSStatus HRGetHTMLURL(
                    HRReference  hrRef
                    Handle       HTMLURLH)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`HTMLURLH`

A handle. Make sure you allocate a valid handle with the `NewHandle` function. On return, this handle (which may be resized) references a C string containing the URL of the given file.

function result

A result code. For a list of possible return values, see Result Codes.

HRGetRootURL

Obtains the root URL for all relative links on a given HTML page.

```
pascal OSStatus HRGetRootURL(
                    HRReference  hrRef
                    Handle       rootURLH)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`rootURLH`

A handle. Make sure you allocate a valid handle with the `NewHandle` function. On return, this handle (which may be resized) references a C string containing the root URL of the given file.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

The root URL is normally the URL of the page specified in the `hrRef` parameter, unless the page contains a `<BASE>` tag specifying a different root URL.

HRGetBaseURL

Obtains the base URL of a given HTML page.

```
pascal OSStatus HRGetBaseURL(
                    HRReference  hrRef
                    Handle       baseURLH)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`baseURLH`

A handle. Make sure you allocate a valid handle with the `NewHandle` function. On return, this handle (which may be resized) references a C string containing the URL of the given file. If the current page does not contain a `<BASE>` tag, the handle is empty.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

The URL returned by this function begins with `'file:///'`.

HRGetTitle

Obtains the title of a given HTML page.

```
pascal OSStatus HRGetTitle(
                    HRReference hrRef,
                    StringPtr title)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`title`

A `StringPtr`. On return, the HTML Rendering Library stores the contents of the `<TITLE>` tag of the given page in the referenced string.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

This function converts any HTML entities into Mac OS characters before storing them in the string specified in the `title` parameter.

HRGetHTMLFile

Obtains a file system specification record for a given HTML page.

```
pascal OSStatus HRGetHTMLFile(
                    HRReference hrRef,
                    FSSpec      *fsspec)
```

`hrRef`

A `HRReference` value previously obtained by your application.

`fsspec`

A pointer to a file system specification record (`FSSpec`). On return, the `FSSpec` contains file specification data for the page specified in the `hrRef` parameter.

function result

A result code. For a list of possible return values, see Result Codes.

Converting URL and FSSpec Data

The HTML Rendering Library provides the following functions to convert `URL` and `FSSpec` data:

- `HRUtilCreateFullURL`
- `HRUtilGetFSSpecFromURL`
- `HRUtilGetURLFromFSSpec`

HRUtilCreateFullURL

Obtains a full URL from a given set of relative URLs.

```
pascal OSStatus HRUtilCreateFullURL(
                    const char   *rootURL,
                    const char   *linkURL
                    Handle       fullURLH)
```

`rootURL`

A pointer to a C string containing the root URL that the HTML Rendering Library will use to create the full URL. The root URL typically identifies an HTML source page.

`linkURL`

A pointer to a C string containing the link URL that the HTML Rendering Library will use to create the full URL. The link URL typically identifies a link being clicked on.

`fullURLH`

A handle. On return, this handle references a full URL created from the specified root url and the specified link URL. You must allocate this handle with the function `NewHandle()` before calling `HRUtilCreateFullURL`. The HTML Rendering Library resizes the handle and terminates the C string with a `NULL` character.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

This function allows you to obtain a full URL to use when a relative URL is inappropriate. This might be useful when you need to pass a URL to another application, for example.

HRUtilGetFSSpecFromURL

Obtains a `FSSpec` from a given set of relative URL's.

```
pascal OSStatus HRUtilGetFSSpecFromURL(
                    const char   *rootURL,
                    const char   *linkURL
                    FSSpec       *destSpec)
```

`rootURL`

A pointer to a C string containing the root URL that the HTML Rendering Library will use to create the full URL. The root URL typically identifies an HTML source page.

`linkURL`

A pointer to a C string containing the link URL that the HTML Rendering Library will use to create the full URL. The link URL typically identifies a link being clicked on.

`destSpec`

A pointer to a file system specification record (`FSSpec`). On return, this points to the `FSSPec` of the full URL calculated from the specified root url and the specified link URL.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

This function allows you to obtain a `FSSpec` to use when a relative URL is inappropriate. This might be useful for Apple events, for example.

HRUtilGetURLFromFSSpec

Obtains a full URL from a given `FSSpec`.

```
pascal OSStatus HRUtilGetURLFromFSSpec(
                    const FSSpec *fsspec)
                    Handle      urlHandle
```

`fsspec`

A pointer to an existing file system specification record (`FSSpec`).

`urlHandle`

A handle to a C string. On return, this C string contains the URL of the given `FSSpec`. You must allocate this handle with the function `NewHandle()` before calling `HRUtilGetURLFromFSSpec`. The HTML Rendering Library resizes the handle and terminates the C string with a `NULL` character.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

This function allows you to obtain a URL from a given `FSSpec`. This might be useful when you have previously obtained an `FSSpec` and need to pass it to an application that requires URL data, for example.

Application-Defined Functions

The HTML Rendering Library does not provide any link-tracking services; it is strictly intended to handle rendering. Therefore, it provides for three types of application-defined functions:

1. A function to keep track of newly visited links. This is described in Handling Newly Visited Links.

2. A function to keep track of previously visited links. This is described in Handling Previously Visited Links.

3. A function to intercept and redirect URL's. This is described in Intercepting and Redirecting URL's.

Handling Newly Visited Links

MyNewURLProc

You can provide an application-defined function that keeps track of newly visited links, which is useful for maintaining a history list, for example. Below is how you would define such a function if you chose to call it `MyNewURLProc`:

```
OSStatus MyNewURLProc(
                const char   *url
                const char   *targetFrame,
                Boolean      addToHistory,
                void         *refCon
```

`url`

A pointer to a C string containing the URL of the link.

`targetFrame`

A pointer to a C string containing the name of the target frame.

`addToHistory`

The HTML Rendering Library passes `true` in this parameter to indicate that you should add this link to a link-tracking history. It is up to your application to do any link-tracking.

`refCon`

An arbitrary value set by your application. This value is passed by your application when you call the `HRRegisterNewURLUPP` function and passed back when the HTML Rendering Library calls your `MyNewURLProc` function. You may find this value useful for referring to an object instance or a structure, for example.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

The sequence of steps required to implement an application-defined function to track visited links is as follows:

1. Obtain a UPP for your application-defined function by calling the `NewHRNewURLUPP` function.

2. Register your application-defined function by passing the UPP to the `HRRegisterNewURLUPP` function.

3. When the HTML Rendering Library calls your application-defined function, take note of the URL being visited.

When you are done using your application-defined function:

1. Unregister your application-defined function by calling the `HRUnregisterNewURLUPP` function.

2. Dispose of the UPP by calling the `DisposeHRNewURLUPP` function.

NewHRNewURLUPP

Obtains a UPP for an application-defined function that handles newly visited links.

```
pascal NewHRNewURLUPP NewHRNewURLUPP(HRNewURLProcPtr userRoutine)
```

`userRoutine`

A pointer to your application-defined function that handles newly visited links. For more information, see `MyNewURLProc`.

function result

A Universal Procedure Pointer. You pass this pointer to the `HRRegisterNewURLUPP` function.

HRRegisterNewURLUPP

Registers an application-defined function that handles newly visited links.

```
pascal void HRRegisterNewURLUPP(
                HRNewURLUPP  inNewURLUPP,
```

```
                HRReference  hrRef,
                void         *inRefCon)
```

`inNewURLUPP`

A Universal Procedure Pointer (UPP). You obtain this UPP by calling the `NewHRNewURLUPP` function.

`hrRef`

A `HRReference` value previously obtained by your application.

`inRefCon`

An arbitrary value set by your application. This value is passed by your application when you call the `HRRegisterNewURLUPP` function and passed back when the HTML Rendering Library calls your `MyNewURLProc` function. You may find this value useful for referring to an object instance or a structure, for example.

HRUnregisterNewURLUPP

Unregisters a previously registered application-defined function.

```
pascal void HRUnregisterNewURLUPP(HRReference hrRef)
```

`hrRef`

A `HRReference` value. You pass the same `HRReference` value that you passed to register the application-defined function.

DisposeHRNewURLUPP

Disposes of a previously obtained UPP.

```
pascal void DisposeHRNewURLUPP(HRNewURLUPP userUPP)
```

`userUPP`

A Universal Procedure Pointer (UPP) that you previously obtained by calling the `NewHRNewURLUPP` function.

MyHRWasURLVisitedProc

You can provide an application-defined function that keeps track of whether a given URL has been previously visited. Below is how you would define such a function if you chose to call it `MyHRWasURLVisitedProc`:

```
Boolean MyHRWasURLVisitedProc(
                const char          *url
                void                *refCon)
```

`url`

A pointer to a C string containing the URL of the link.

`refCon`

An arbitrary value set by your application. This value is passed by your application when you call the `HRRegisterWasURLVisitedUPP` function and passed back when the HTML Rendering Library calls your `MyHRWasURLVisitedProc` function. You may find this value useful for referring to an object instance or a structure, for example.

function result

If the given URL was previously visited, your application-defined function should return `true`. If the given URL was not previously visited, your application-defined function should return `false`.

DISCUSSION

The sequence of steps required to implement an application-defined function to handle previously visited links is as follows:

1. Obtain a UPP for your application-defined function by calling the `NewHRWasURLVisitedUPP` function.

2. Register your application-defined function by passing the UPP to the `HRRegisterWasURLVisitedUPP` function.

3. Respond when the HTML Rendering Library calls your application-defined function.

When you are done using your application-defined function:

1. Unregister your application-defined function by calling the `HRUnregisterWasURLVisitedUPP` function.

2. Dispose of the UPP by calling the `DisposeHRWasURLVisitedUPP` function.

NewHRWasURLVisitedUPP

Obtains a UPP for an application-defined function that handles previously visited links.

```
pascal NewHRWasURLVisitedUPP NewHRWasURLVisitedUPP(
                                HRWasURLVisitedProcPtr userRoutine)
```

`userRoutine`

A pointer to your application-defined function that handles visited links. For more information, see `MyHRWasURLVisitedProc`.

function result

A Universal Procedure Pointer. You pass this pointer to the `HRRegisterWasURLVisitedUPP` function.

HRRegisterWasURLVisitedUPP

Registers an application-defined function that handles previously visited links.

```
pascal void HRRegisterWasURLVisitedUPP(
                HRWasURLVisitedUPP  inWasURLVisitedUPP,
                HRReference          hrRef,
                void                 *inRefCon)
```

`inWasURLVisitedUPP`

A Universal Procedure Pointer (UPP). You obtain this UPP by calling the `NewHRWasURLVisitedUPP` function.

`hrRef`

A `HRReference` value previously obtained by your application.

`inRefCon`

An arbitrary value set by your application. This value is passed by your application when you call the `HRRegisterWasURLVisitedUPP` function and passed back when the HTML Rendering Library calls your `MyHRWasURLVisitedProc` function. You may find this value useful for referring to an object instance or a structure, for example.

HRUnregisterWasURLVisitedUPP

Unregisters a previously registered application-defined function.

```
pascal void HRUnregisterWasURLVisitedUPP(HRReference hrRef)
```

`hrRef`

A `HRReference` value. You pass the same `HRReference` value that you passed to register the application-defined function.

DisposeHRWasURLVisitedUPP

Disposes of a previously obtained UPP.

```
pascal void DisposeHRWasURLVisitedUPP(HRWasURLVisitedUPP userUPP)
```

`userUPP`

A Universal Procedure Pointer (UPP) that you previously obtained by calling the `NewHRWasURLVisitedUPP` function.

Intercepting and Redirecting URL's

MyHRURLToFSSpecProc

You can provide an application-defined function that converts URL data to `FSSpec` data. This is useful if you want to redirect certain URL's to application-specific files, for example. Below is how you would define such a function if you chose to call it `MyHRURLToFSSpecProc`:

```
OSStatus MyHRURLToFSSpecProc(
                const char      *rootURL,
                const char      *linkURL,
                FSSpec          *fsspec,
                URLSourceType   urlSourceType,
                void            *refCon)
```

`rootURL`

A pointer to a C string containing the root URL of the file to be loaded.

`linkURL`

A pointer to a C string containing the link URL of the file to be loaded.

`fsspec`

A pointer to a file system specification record (`FSSpec`) that you use to specify the file to which the HTML Rendering Library redirects the user.

`urlSourceType`

The HTML Rendering Library passes one of the following constants to indicate the type of file being searched for:

- `kHRLookingForHTMLSource` indicates that the file is an HTML source document
- `kHRLookingForImage` indicates that the file is an image
- `kHRLookingForEmbedded` indicates that the file is an embedded object, such as a QuickTime movie
- `kHRLookingForImageMap` indicates that the file is an HTML image map
- `kHRLookingForFrame` indicates that the file is an HTML frameset

`refCon`

An arbitrary value set by your application. This value is passed by your application when you call the `HRRegisterURLToFSSpecUPP` function and passed back when the HTML Rendering Library calls your `MyHRURLToFSSpecProc` function. You may find this value useful for referring to an object instance or a structure, for example.

function result

A result code. For a list of possible return values, see Result Codes.

DISCUSSION

The sequence of steps required to implement an application-defined function to intercept URL's is as follows:

1. Obtain a UPP for your application-defined function by calling the `NewHRURLToFSSpecUPP` function.

2. Register your application-defined function by passing the UPP to the `HRRegisterURLToFSSpecUPP` function.

3. Respond when the HTML Rendering Library calls your application-defined function.

When you are done with your application-defined function:

1. Unregister your application-defined function by calling the `HRUnregisterURLToFSSpecUPP` function.

2. Dispose of the UPP by calling the `DisposeHRURLToFSSpecUPP` function.

NewHRURLToFSSpecUPP

Obtains a UPP for an application-defined function that intercepts URL's.

```
pascal HRWasURLVisitedUPP NewHRWasURLVisitedUPP(
                            HRWasURLVisitedProcPtr userRoutine)
```

`userRoutine`

A pointer to your application-defined function that intercepts URL's. For more information, see `MyHRURLToFSSpecProc`.

function result

A Universal Procedure Pointer. You pass this pointer to the `HRRegisterURLToFSSpecUPP` function.

HRRegisterURLToFSSpecUPP

Registers an application-defined function that handles previously visited links.

```
pascal void HRRegisterURLToFSSpecUPP(
                HRURLToFSSpecUPP inURLToFSSpecUPP,
                HRReference      hrRef,
                void             *inRefCon)
```

`inURLToFSSpecUPP`

A Universal Procedure Pointer (UPP). You obtain this UPP by calling the `NewHRURLToFSSpecUPP` function.

`hrRef`

A `HRReference` value previously obtained by your application.

`inRefCon`

An arbitrary value set by your application. This value is passed by your application when you call the `HRRegisterURLToFSSpecUPP` function and passed back when the HTML Rendering Library calls your `MyHRURLToFSSpecProc` function. You may find this value useful for referring to an object instance or a structure, for example.

HRUnregisterURLToFSSpecUPP

Unregisters a previously registered application-defined function.

```
pascal void HRUnregisterURLToFSSpecUPP(HRReference hrRef)
```

`hrRef`

A `HRReference` value. You pass the same `HRReference` value that you passed to register the application-defined function.

DisposeHRURLToFSSpecUPP

Disposes of a previously obtained UPP.

```
pascal void DisposeHRWasURLVisitedUPP(HRWasURLVisitedUPP userUPP)
```

`userUPP`

A Universal Procedure Pointer (UPP) that you previously obtained by calling the `NewHRURLToFSSpecUPP` function.

HTML Rendering Library Constants

Scrollbar State Constants

The `HRScrollbarState` enumeration defines constants you can pass to the `HRSetScrollbarState` function to specify whether scrollbars are drawn in the rendering area.

```
typedef SInt16   HRScrollbarState;
enum {
    eHRScrollbarOn   = 0,
    eHRScrollbarOff  = 1,
    eHRScrollbarAuto = 2
};
```

- `eHRScrollbarOn` tells the HTML Rendering Library to draw scrollbars at all times. If the data does not fill the current view, the scroll bars are inactive.
- `eHRScrollbarOff` tells the HTML Rendering Library never to draw scrollbars. You may find this option useful if if the HTML rendering area you specify does not extend to the edge of a window.
- `eHRScrollbarAuto` tells the HTML Rendering Library to draw scrollbars as needed. This is the default setting.

URL Source Type Constants

The `URLSourceType` enumeration defines constants the HTML Rendering Library passes in the `urlSourceType` parameter of your application-defined function that redirects URL's to file system specification records, as described in MyHRURLToFSSpecProc. The constants describe the type of file that your application-defined function is being asked to evaluate.

```
typedef UInt16   URLSourceType;

enum {
    kHRLookingForHTMLSource  = 1,
```

```
    kHRLookingForImage      = 2,
    kHRLookingForEmbedded   = 3,
    kHRLookingForImageMap   = 4,
    kHRLookingForFrame      = 5
};
```

- `kHRLookingForHTMLSource` indicates that the file is an HTML source document.
- `kHRLookingForImage` indicates that the file is an image.
- `kHRLookingForEmbedded` indicates that the file is an embedded object, such as a QuickTime movie.
- `kHRLookingForImageMap` indicates that the file is an HTML image map.
- `kHRLookingForFrame` indicates that the file is an HTML frameset.

Result Codes

The HTML Rendering library can return result codes from other managers, such as the File Manager, in addition to the result codes listed below.

Table 1 Result codes for HTML Rendering Library

0	noErr	No error has occurred.
-50	paramErr	Unrecognized `HRReference`.
-5360	kHRHTMLRenderingLibNotInstalledErr	The HTML Rendering Library is not available.
-5361	kHRMiscellaneousExceptionErr	An unexpected exception occurred.
-5362	kHRUnableToResizeHandleErr	Unable to resize a handle to the required size.

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. HTML processing was done with BBEdit. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER

Otto Schlosser

LEAD ENGINEER

Riley Howard

PRODUCTION EDITOR

Lorraine Findlay

ACKNOWLEDGMENTS

Tony Francis, Rick Hoiberg, Gordon Meyer, James Miyake