## Contents

### Preface
- Description ................................................. 7
- Contents .................................................. 7
- Other Useful Documentation ............................. 7
- Conventions Used in This Book ............................ 8

### Overview
- Description .................................................. 11
- Weblink Services ........................................... 11
- Weblink Driver ............................................. 11
  - Writing a Custom Driver ................................. 12
  - Sequence of Actions .................................... 12

### Weblink Methods
- BeginWebProgress ......................................... 15
- EndWebProgress ........................................... 16
- GetAppSpecifier ............................................ 17
- RegisterWebDriver ......................................... 18
- UpdateWebProgress ........................................ 19
- WebProgressDidCancel .................................... 20
- WLCreateOrDeleteWebLinks ............................... 21
- WWWOpenURL ............................................. 22
- WWWOpenURLWithParams ................................. 23

### Weblink Callbacks
- WDAppSupportPredicate .................................. 25
- WDBecomeDriverFunction ................................ 26
- WDBringToFrontFunction .................................. 27
- WDFollowLinkFunction ..................................... 28
- WDFollowLinkWithParamsFunction ....................... 29
- WDIsDriverBusyFunction .................................. 30
- WDOptionsFunction ........................................ 31
Contents

Weblink Declarations. ................................................. 33

WDAppSpecifier ......................................................... 33
WebDriverVector ....................................................... 34
WebProgressMonitor .................................................... 36
WWWOpenURLParams .................................................. 37
WWWOpenURLParamsRec .............................................. 37
Preface

Description

A link in a PDF document that references a URL is referred to as a Weblink.

The Adobe® Acrobat® Weblink plug-in adds the capability of linking to documents on the World Wide Web. It allows other plug-ins to access its functionality by means of a Host Function Table (HFT) API.

This document describes the Weblink APIs. It also provides information on Adobe’s Standard Weblink Driver, which performs low-level, platform-specific functions with a Web browser on behalf of Weblink.

If you received this technical note without obtaining the entire Acrobat Software Development Kit (SDK), you can get the complete SDK by visiting:


Contents

This technical note contains the following sections:

● Overview describes the Weblink plug-in, the standard driver, and the requirements for writing a custom driver.
● Weblink Methods describes each method, including its parameters and return value.
● Weblink Callbacks describes each callback function, including its parameters and return value.
● Weblink Declarations describes the Weblink data types.

Other Useful Documentation

The Acrobat SDK includes many other books that you might find useful. When mentioned in this document, those books will often appear as live links (blue italic links). However, in order to actually jump from this document to those books, those books must exist in the proper directories within your computer’s file system. This happens automatically when you install the SDK onto your system.

If for some reason you did not install the entire SDK onto your system and you do not have all of the documentation, please visit the Adobe Solutions Network web site (http://partners.adobe.com/asn/) to find the books you need. Then download them and
install them in the proper directories, which can be determined by looking at the Acrobat SDK Documentation Roadmap, included at the beginning of each book in the SDK.

You should be familiar with the Acrobat Core API and Portable Document Format (PDF). The following technical notes provide this information.

**Acrobat Core API Overview:** Gives an overview of the objects and methods provided by the Acrobat core API.

**Acrobat Core API Reference:** Describes in detail the objects and methods provided by the Acrobat core API.

**PDF Reference,** Third Edition, Version 1.4. Provides a description of the PDF file format, as well as suggestions for producing efficient PDF files.

### Conventions Used in This Book

The Acrobat documentation uses text styles according to the following conventions.

<table>
<thead>
<tr>
<th>Font</th>
<th>Used for</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>monospaced</td>
<td>Paths and filenames</td>
<td>C:\templates\mytmpl.fm</td>
</tr>
<tr>
<td></td>
<td>Code examples set off from plain text</td>
<td>These are variable declarations: AVMenu commandMenu, helpMenu;</td>
</tr>
<tr>
<td>monospaced bold</td>
<td>Code items within plain text</td>
<td>The GetExtensionID method ...</td>
</tr>
<tr>
<td></td>
<td>Parameter names and literal values in</td>
<td>The enumeration terminates if proc returns false.</td>
</tr>
<tr>
<td></td>
<td>reference documents</td>
<td></td>
</tr>
<tr>
<td>monospaced italic</td>
<td>Pseudocode</td>
<td>ACCB1 void ACCB2 ExeProc(void) { do something }</td>
</tr>
<tr>
<td></td>
<td>Placeholders in code examples</td>
<td>AFSimple_Calculate(cFunction,cFields)</td>
</tr>
<tr>
<td>blue</td>
<td>Live links to Web pages</td>
<td>The Acrobat Solutions Network URL is: <a href="http://partners/adobe.com/asn/">http://partners/adobe.com/asn/</a></td>
</tr>
<tr>
<td></td>
<td>Live links to sections within this document</td>
<td>See Using the SDK.</td>
</tr>
<tr>
<td></td>
<td>Live links to other Acrobat SDK documents</td>
<td>See the Acrobat Core API Overview.</td>
</tr>
<tr>
<td></td>
<td>Live links to code items within this document</td>
<td>Test whether an ASAtom exists.</td>
</tr>
<tr>
<td><strong>Font</strong></td>
<td><strong>Used for</strong></td>
<td><strong>Examples</strong></td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>bold</strong></td>
<td>PostScript language and PDF operators, keywords, dictionary key names</td>
<td>The <code>setpagedevice</code> operator</td>
</tr>
<tr>
<td></td>
<td>User interface names</td>
<td>The File menu</td>
</tr>
<tr>
<td><strong>italic</strong></td>
<td>Document titles that are not live links</td>
<td>Acrobat Core API Overview</td>
</tr>
<tr>
<td></td>
<td>New terms</td>
<td>User space specifies coordinates for...</td>
</tr>
<tr>
<td></td>
<td>PostScript variables</td>
<td><code>filename deletefile</code></td>
</tr>
</tbody>
</table>
Overview

Description

The Acrobat Weblink plug-in exports its own Host Function Table (HFT), whose methods can be used by other plug-ins. The HFT’s name is defined in the `WLHFTNAME` macro, and its version number is `WEB_LINK_HFT_LATEST_VERSION`.

To use the Weblink plug-in’s HFT, a plug-in must include the header file `WeblinkHFT.h`. The plug-in must also import the HFT using `ASExtensionMgrGetHFT` and assign the HFT returned by this call to a plug-in-defined global variable named `gWLHFT`. The easiest way to do this is to use the `Init_gWLHFT` macro defined in the header files.

Weblink Services

The Weblink plug-in provides the following services:

- Maintenance of links (editing and storage of URLs associated with links, and so on)
- Manipulation of links (appropriate cursor changes and dynamic display of URL destinations)
- Selection of the external Web browser
- Driver manipulation
- Basic progress status services (progress monitor, wait cursor, and so on)

Weblink Driver

The Weblink plug-in includes a standard driver, known as Adobe’s Standard Web Driver. It allows support for transport mechanisms or Web browsers to be added at a later time.

The Standard Driver uses Apple events and DDE messages to communicate with a Web browser. It supports a protocol that consists of a suite of verbs—some going to and some coming from—the Web browser. These verb definitions are provided so that Web browsers can implement this protocol to be compatible with the Adobe Standard Web Driver. Each verb is specified in terms of the platform-specific implementation: Apple events for the Macintosh and DDE for Windows. The Standard Driver’s use of each verb is also described. Browsers that wish to use their own protocol may do so by writing a custom driver.

The Weblink plug-in communications software in the Weblink driver is independent of the Acrobat mechanism for handling links (the PDF implementation of URLs). This separation improves portability by isolating the highly platform-specific Interapplication
Communication messages. Even on a given platform, there is no standard among Web browsers for handling interapplication communication, and the actual transport mechanism may vary over time. By separating out the transport code, the Weblink plug-in remains portable across platforms, across different vendors’ implementations of Web browsers, and across different versions of Web browsers from the same vendor.

Writing a Custom Driver

A driver is an Acrobat core plug-in, written like any other plug-in. A driver must register itself with the Weblink plug-in during the import, replace, and register phase of the plug-in initialization process by calling `RegisterWebDriver`. You pass this method a `WebDriverVector` structure containing a version number and six pointers to functions that your driver provides to handle Web-browser-specific tasks.

A driver is responsible for the following:

- Connection with external services (either directly or through an external application)
- Communication with external services
- Association of a base URL with a given document
- Identification of external browsers that are compatible with the driver

Sequence of Actions

In a typical session, the following actions might occur.

1. The user launches Acrobat.
   a) The Weblink plug-in publishes a Host Function Table (HFT) during the `exportHFTsCallback` phase of initialization.
   b) During the `importReplaceAndRegisterCallback` phase, all drivers in turn call `RegisterWebDriver` in the Weblink plug-in’s HFT to register themselves as available.
   c) During the `initCallback` phase, the Weblink plug-in, if possible, selects an appropriate driver and notifies it that it is the active driver.

2. The user opens a PDF document with Weblinks and clicks a Weblink.
   a) The Weblink plug-in extracts the URL from the link and passes it to the driver.
   b) The driver packages the URL into an Interapplication Communication (IAC) message and sends it to an external Web browser (launching the Web browser application, if necessary).
   c) The external Web browser brings itself to the foreground unless the URL’s MIME type is `application/pdf`.
3. The Web browser retrieves the document and packages an IAC message.
   a) The driver accepts the IAC message from the browser and opens the PDF document using `AVDocOpenFromFile`. The driver should associate the URL with the document.
   b) To resolve relative links, Weblink prepends either a base URL with the document, or if there is no base URL, the appropriate portion of the URL of the document the link is in. For mapped links—links with the `isMap` attribute—it appends a “?” followed by the mouse location’s x- and y-coordinates.
Weblink Methods

BeginWebProgress

WebProgressMonitor BeginWebProgress (char* message, ASUns32 flags, ASInt32 timeoutTime);

Description

Creates a new Weblink progress monitor to provide feedback during long operations. Current Web browsers follow links asynchronously and use BeginWebProgress, UpdateWebProgress, and EndWebProgress to assure a user that a request is making progress.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message text. A NULL-terminated C string that represents a message to display with the progress monitor.</td>
</tr>
</tbody>
</table>
| flags     | Possible flag options for the progress monitor. There are two flags defined (in WLHft.h):
|           | PROGRESS_HAS_METER — Put a thermometer-style meter in the progress display. |
|           | PROGRESS_HAS_CANCEL — Add either a Cancel button or a message directing the user how to abort the operation. |
| timeoutTime | Timeout period. The amount of time in seconds before the progress monitor indicates that progress is not being made. |

Return Value

Monitor creation status. A non-NULL value if the monitor’s creation was successful, NULL otherwise. In the current implementation, there can be at most one progress monitor.
EndWebProgress

    void EndWebProgress (WebProgressMonitor progMon);

Description
Terminates the current progress monitor.

Parameters

| progMon | Current progress monitor. |

Return Value
None
**GetAppSpecifier**

```java
WDAppSpecifier GetAppSpecifier (boolean promptUser);
```

**Description**

Obtains a platform-specific structure specifying the application the user has chosen as the Web browser.

**Parameters**

| promptUser | true if there is no current Web browser and the user is prompted to choose one, false otherwise. |

**Return Value**

Pointer to a platform-specific data structure that specifies the Web browser. On the Mac OS, it is a pointer to an `FSSpec` data structure. On Windows and UNIX platforms, it is a pointer to a string (`char*`) representing the full path of the executable application.
RegisterWebDriver

ASBool RegisterWebDriver (char* driverName, void* driverRock, WebDriverVector* wdVec);

Description

Registers a driver with the Weblink plug-in. Must be called during the import, replace, and register phase of the plug-in initialization process of any plug-ins that want to use Weblink services.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>driverName</td>
<td>The driver’s name. A NULL-terminated C string shown to the user to identify the current driver. The string is limited to 255 characters and is copied internally.</td>
</tr>
<tr>
<td>driverRock</td>
<td>Pointer to a driver-defined data structure. This value is passed to the driver for every driver-supplied function call.</td>
</tr>
<tr>
<td>wdVec</td>
<td>Pointer to a data structure of driver-supplied callback functions. It is not copied; therefore, the structure must be either statically or dynamically allocated—rather than automatically allocated as a local variable. It must not be modified, moved, or deallocated until the plug-in terminates within Acrobat.</td>
</tr>
</tbody>
</table>

Return Value

Registration status. true if the registration was successful, false otherwise. Failure occurs if there is insufficient memory or the version number in wdVec is incompatible with the Weblink plug-in.
UpdateWebProgress

ASBool UpdateWebProgress (WebProgressMonitor progMon,
char* message, ASInt32 from, ASInt32 to, ASInt32 outOf,
ASInt32 timeoutTime);

Description

Updates the state of the current progress monitor.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>progMon</td>
<td>Current progress monitor returned from BeginWebProgress.</td>
</tr>
<tr>
<td>message</td>
<td>Message text. If non-NULL, the text displayed is changed to that represented by message. If message is NULL, the text is not changed.</td>
</tr>
<tr>
<td>from</td>
<td>Lower bound of a subrange between 0 and outOf to highlight: from and to are the lower and upper bounds of a subrange between 0 and outOf to highlight. Typically the lower bound, from, is zero, but a driver may wish to indicate progress of an operation (that has no magnitude known) as a priority. It could do so with a “chaser light” display by cycling the lower bound, from, through the range between 0 and outOf - 1, and defining the upper bound, to, as from + 1.</td>
</tr>
<tr>
<td>to</td>
<td>Upper bound of subrange between 0 and outOf to highlight.</td>
</tr>
<tr>
<td>outOf</td>
<td>If progMon was created with the PROGRESS_HAS_METER flag, the from to, and outOf arguments are used to change the state of the bar. The range of the meter is defined from 0 to outOf.</td>
</tr>
<tr>
<td>timeoutTime</td>
<td>(Optional) New timeout period in seconds. If non-zero, a new timeout is set.</td>
</tr>
</tbody>
</table>

Return Value

Update status. true if the update was successful, false if the monitor timed out or the user canceled the operation. The latter case only happens when the progress monitor was created with the PROGRESS_HAS_CANCEL flag in BeginWebProgress.
WebProgressDidCancel

ASBool WebProgressDidCancel (WebProgressMonitor progMon);

Description
Tests whether or not the user clicked the Cancel button in the progress monitor dialog.

Parameters

progMon Current progress monitor.

Return Value
true if the progress monitor has been canceled, false otherwise.
WLCreateOrDeleteWebLinks

ASInt32 WLCreateOrDeleteWebLinks (ASBool bCreate, PDDoc pdd, ASInt32 nStart, ASInt32 nEnd, ProgressMonitor pm, void *vMData, CancelProc cp, void *vCPData);

Description

Used by other plug-ins that want to create or delete weblinks in a document. Creates or deletes all web links in the specified document within the specified page range. Displays a progress monitor during the operation, which allows the user to cancel.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bCreate</td>
<td>When true, create a weblink. When false, delete a weblink.</td>
</tr>
<tr>
<td>pdd</td>
<td>The document in which to create or delete the weblink.</td>
</tr>
<tr>
<td>nStart</td>
<td>The starting page number.</td>
</tr>
<tr>
<td>nEnd</td>
<td>The ending page number.</td>
</tr>
<tr>
<td>pm</td>
<td>The current progress monitor.</td>
</tr>
<tr>
<td>vMData</td>
<td>A pointer to data to be passed to the progress monitor.</td>
</tr>
<tr>
<td>cp</td>
<td>A procedure to call when if the user cancels the operation.</td>
</tr>
<tr>
<td>vCPData</td>
<td>Client-defined data to pass to the cancellation procedure.</td>
</tr>
</tbody>
</table>

Return Value

The number of weblinks created or deleted.
WWWOpenURL

ASBool WWWOpenURL (AVDoc avd, char* cURL, char* cFormData);

Description
Asks Weblink to follow the specified URL. If the URL specifies a PDF document, opens it in a new window (an AVDoc view).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avd</td>
<td>If specified, the AVDoc that contains the source of the URL.</td>
</tr>
<tr>
<td>cURL</td>
<td>The URL to be opened.</td>
</tr>
<tr>
<td>cFormData</td>
<td>Data for a form to be posted.</td>
</tr>
</tbody>
</table>

Return Value
true if the request succeeded, false otherwise.

Example

```c
#include "WLHft.h"

HPT gWLHFT = Init_gWLHFT;
if (!gWLHFT)
    return false;
WWWOpenURL(AVAppGetActiveDoc(), "http://www.adobe.com", 0);```

WWWOpenURLWithParams

ASBool WWWOpenURL (AVDoc avd, char* cURL, char* cFormData, WWWOpenURLParams params);

Description
Asks Weblink to follow a URL, specifying a target frame. If the URL specifies a PDF document, opens it in a new window (an AVDoc view).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avd</td>
<td>If specified, the AVDoc that contains the source of the URL.</td>
</tr>
<tr>
<td>cURL</td>
<td>The URL to be opened.</td>
</tr>
<tr>
<td>cFormData</td>
<td>Data for a form to be posted.</td>
</tr>
<tr>
<td>params</td>
<td>Parameter block specifying name of frame.</td>
</tr>
</tbody>
</table>

Return Value

true if the request succeeded, false otherwise.
**WDAppSupportPredicate**

```
ASBool WDAppSupportPredicate (void* driverRock, WDAppSpecifier theApp);
```

**Description**

Tests whether or not a driver can support a given Web browser application.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>driverRock</code></td>
<td>Pointer to a driver-defined data structure provided in <code>RegisterWebDriver</code>.</td>
</tr>
<tr>
<td><code>WDAppSpecifier</code></td>
<td>Platform-specific data structure used to identify the Web browser application. On the Mac OS, it is a pointer to an <code>FSSpec</code> data structure. On Windows, it is a pointer to a string (char*) representing the full path of the executable application.</td>
</tr>
</tbody>
</table>

**Return Value**

True if the driver is compatible with the given Web browser application, false otherwise.
WDBecomeDriverFunction

ASBool WDBecomeDriverFunction (void* driverRock, ASBool bComing);

Description
Called whenever the state of the driver changes from inactive to current, or vice versa.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>driverRock</td>
<td>Pointer to a driver-defined data structure provided in RegisterWebDriver.</td>
</tr>
<tr>
<td>bComing</td>
<td>Tells the driver whether it should become current or relinquish control. If true, the driver is being requested to become the current Weblink driver, and it should take any action that is appropriate, such as installing IAC message handlers, opening TCP drivers, and so on. If false, the driver is being asked to relinquish its status as current driver and should take whatever action is necessary to accomplish this.</td>
</tr>
</tbody>
</table>

Return Value
false if the driver fails to change state to that requested in bComing, otherwise true.
**WDBringFrontFunction**

```c
void* WDBringFrontFunction (void* driverRock);
```

**Description**

Called when the user clicks on the Weblink toolbar button to switch to the browser application.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>driverRock</code></td>
<td>Pointer to a driver-defined data structure provided in <code>RegisterWebDriver</code>.</td>
</tr>
</tbody>
</table>

**Return Value**

None
WDFollowLinkFunction

ASBool WDFollowLinkFunction (void* driverRock, AVDoc theDoc, char* URL, char* cFormData);

Description

Follows and retrieves the link specified by a URL from the PDF document specified by theDoc. Weblink supplies the driver a fully resolved the URL. The driver is solely responsible for launching a Web browser, if it requires one. If the driver depends on an external Web browser, the driver or the Web browser is responsible for bringing that browser to the foreground if the link data is not displayed by the Acrobat viewer.

For relative links, Weblink prepends a base URL if specified; otherwise, it prepends the appropriate portion from the current document’s URL to resolve a relative link.

Weblink automatically handles mapped links, that is, links that generate different results based on the location of the mouse within the links. (Such links are specified by checking the “Mapped Coordinates Link” box in the Edit URL dialog box. These links have the isMap attribute.) Weblink appends “?” to the URL, followed by the x- and y-coordinates of the mouse at the time the link was selected, relative to the upper-left corner of the link’s rectangle.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>driverRock</td>
<td>Pointer to a driver-defined data structure provided in RegisterWebDriver.</td>
</tr>
<tr>
<td>theDoc</td>
<td>The PDF document that contains the link.</td>
</tr>
<tr>
<td>URL</td>
<td>A NULL-terminated C string containing the URL the link points to.</td>
</tr>
<tr>
<td>cFormData</td>
<td>Data from a form to be sent with the URL as post data. The default MIME type of form data is “application/x-www-form-unencoded”.</td>
</tr>
</tbody>
</table>

Return Value

true if the driver has successfully begun the process of following the link, false otherwise.
**WDFollowLinkWithParamsFunction**

```c
ASBool WDFollowLinkFunction (void* driverRock, AVDoc theDoc, char* cURL, char* cFormData, WWWOpenURLParams params);
```

**Description**

Follows and retrieves the link specified by a URL from the PDF document specified by `theDoc`, going to a target frame. Weblink supplies the driver a fully resolved the URL. The driver is solely responsible for launching a Web browser, if it requires one. If the driver depends on an external Web browser, the driver or the Web browser is responsible for bringing that browser to the foreground if the link data is not displayed by the Acrobat viewer.

For relative links, Weblink prepends a base URL if specified; otherwise, it prepends the appropriate portion from the current document’s URL to resolve a relative link.

Weblink automatically handles mapped links, that is, links that generate different results based on the location of the mouse within the links. (Such links are specified by checking the “Mapped Coordinates Link” box in the Edit URL dialog box. These links have the `isMap` attribute.) Weblink appends “?” to the URL, followed by the x- and y-coordinates of the mouse at the time the link was selected, relative to the upper-left corner of the link’s rectangle.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>driverRock</code></td>
<td>Pointer to a driver-defined data structure provided in <code>RegisterWebDriver</code>.</td>
</tr>
<tr>
<td><code>theDoc</code></td>
<td>The PDF document that contains the link.</td>
</tr>
<tr>
<td><code>cURL</code></td>
<td>A <code>NULL</code>-terminated C string containing the URL the link points to.</td>
</tr>
<tr>
<td><code>cFormData</code></td>
<td>Data from a form to be sent with the URL as post data.</td>
</tr>
<tr>
<td><code>params</code></td>
<td>Parameter block specifying a target frame.</td>
</tr>
</tbody>
</table>

**Return Value**

`true` if the driver has successfully begun the process of following the link, `false` otherwise.
WDIsDriverBusyFunction

ASBool WDIsDriverBusyFunction (void* driverRock);

Description

(Optional) Called to determine if the driver is performing a transaction or not. Weblink uses this to check that the driver is not busy before allowing a user to change preferences. Busy is driver-defined. The driver should return busy whenever it is not convenient for it to become the current driver as requested by WDBecomeDriverFunction.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>driverRock</td>
<td>Pointer to a driver-defined data structure provided in RegisterWebDriver.</td>
</tr>
</tbody>
</table>

Return Value

true if the driver is busy, false otherwise.
**WDOptionsFunction**

```c
void* WDOptionsFunction (void* driverRock);
```

**Description**

*(Optional)* Called when the user clicks the Options button in the Weblink Preferences dialog box. If this function is **NULL**, the Options button is disabled.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>driverRock</code></td>
<td>Pointer to a driver-defined data structure provided in <a href="#">RegisterWebDriver</a>.</td>
</tr>
</tbody>
</table>

**Return Value**

None
Weblink Declarations

WDAppSpecifier

/* In Mac OS */
typedef FSSpec* WDAppSpecifier;
/* In UNIX and Windows*/
typedef char* WDAppSpecifier;

Description
Data structure that specifies the pathname to the Web browser.

Header File
WLHFT.h

Related Functions
WDAppSupportPredicate

Related Methods
GetAppSpecifier
**WebDriverVector**

typedef struct {
    ASUns32 WDVRev;
    ASUns32 nReserved;
    WDAppSupportPredicate doYouSupport;
    WDFollowLinkFunction followLink;
    WDBecomeDriverFunction becomeDriver;
    WDBringFrontFunction bringFront;
    WDOptionsFunction options;
    WDIssDriverBusyFunction isDriverBusy;
    WDFollowLinkWithParamsFunction followLinkWithParams;
} WebDriverVector;

**Description**
Data structure containing callbacks that implement a Weblink driver.

**Members**

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDVRev</td>
<td>Driver version number.</td>
</tr>
<tr>
<td>nReserved</td>
<td>Currently unused.</td>
</tr>
</tbody>
</table>

**Header File**
WLHFT.h

**Related Methods**
RegisterWebDriver

**Example**

```c
#include "WLCalls.h"
#include "PICommon.h"
#include "ASCalls.h"
#include "PIMain.h"

HFT gWLHFT;
static WebDriverVector myWDV;

ACCB1 boolean ACCB2 RegisterMyDriver(void)
{
    /* try to get the Weblink HFT */
    gWLHFT = Init_gWLHFT;
    if (!gWLHFT) return false;
    /* Set up the Web Driver Vector */
    memset(myWDV, 0, sizeof(myWDV));
    myWDV.WDVRev = WDVCURRENTREV;
    ```
/* set the version */
/* fill in the functions (all are required) */
myWDV.doYouSupport = ASCallbackCreateProto(
    WDAppSupportPredicate,
    &MySupportPredicate);
myWDV.followLink = ASCallbackCreateProto(
    WDFollowLinkFunction, &MyFollowLink);

myWDV.becomeDriver = ASCallbackCreateProto(
    WDBecomeDriverFunction, &MyBecomeDriver);
myWDV.bringFront = ASCallbackCreateProto(
    WDBringFrontFunction, &MyBringFront);
if (!RegisterWebDriver("Cool Web v1.0", NULL, &myWDV))
    return false;
return true;

/* add the following to the PIHandshake routine */
hsData->importReplaceAndRegisterCallback = ASCallbackCreateProto(
    PIImportReplaceAndRegisterProcType, &RegisterMyDriver);
WebProgressMonitor

typedef void* WebProgressMonitor;

Description
Data structure that implements a progress monitor.

Header File
WLHFT.h

Related Methods
- BeginWebProgress
- UpdateWebProgress
- EndWebProgress
- WebProgressDidCancel
- WLCreateOrDeleteWebLinks
WWWOpenURLParams

WWWOpenURLParamsRec

typedef struct {
    ASSize_t size;
    char * cFrame;
} WWWOpenURLParamsRec, *WWWOpenURLParams;

Description

A parameter block specifying the target frame of a URL.

Members

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Should be initialized to sizeof (WWWOpenURLParamsRec).</td>
</tr>
<tr>
<td>cFrame</td>
<td>The name of the frame to open. This is exactly the same as the TARGET attribute of a link in HTML. You can target a frame by a name you have assigned to it. In HTML, TARGET also has four predefined values that can be used as if certain windows and frames already have names without you having to assign them: _blank, _parent, _self, and _top. Weblink supports only _blank and _self.</td>
</tr>
</tbody>
</table>

Header File

WLHFT.h

Related Method

WWWOpenURLWithParams

Related Callback

WDFollowLinkWithParamsFunction