Guide to Creating Maxthon Plugins

For Maxthon Browser v1.5 (and newer)

September 2006

Revision 0.21

Legal Notice

Corporate Headquarters:

Flat D, BLK 1,9/F

Sky Tower, Tower 1

38 Sung Wong Toi Road

To Kwa Wan, KL

Hong Kong

Tel: +852 9187 0526

Fax: +852 3542 7425

E-mail (customer relations): *netanel@maxthon.com*

E-mail (technical support): *develop@maxthon.com*

Visit us at: http://www.maxthon.com

Table of contents

1. About this guide

In this chapter

1.1 Intended audience

The Guide to Creating Maxthon Plugins is intended for anyone wishing to develop or adapt a plugin for the Maxthon tabbed Internet browser. This includes:

- Organizations with an existing plugin for Microsoft's Internet Explorer
- Organizations with an application that would benefit from tighter integration with a browser
- Individuals who have some programming knowledge and want to enhance their browsing experience

It is assumed that all readers have at least a basic understanding of HTML and some previous experience with the Maxthon browser.

1.2 Purpose of this guide

The intention of this guide is to explain all aspects of creating Maxthon plugins. After reading this guide you should have a thorough knowledge of the potential available from Maxthon plugins.

The guide also aims to explain the differences between Maxthon plugins and plugins for Internet Explorer.

1.3 Organization of the guide

Information about Maxthon plugins is contained in the following three chapters:

Chapter 2: *Understanding Maxthon plugins* (on page 7) - Addresses the background of Maxthon plugins and the ways that plugin support in Maxthon differs from that of Internet Explorer.

Chapter 3: *Types of Maxthon plugin* (on page 11) - Explains the different types of Maxthon plugins including script, COM, EXE, sidebar, button, and toolbar.

Chapter 4: *Developing a Maxthon plugin from an existing IE plugin* (on page 13) - This chapter examines the question of whether developers with existing IE plugins should create a new plugin for Maxthon or adapt their current one. Also covered are testing and troubleshooting a new plugin.

Three appendices provide example plugins with code and screenshots:

Appendix A: Example of a script button plugin (on page 20)

Appendix B: Example of a script sidebar plugin (on page 24)

Appendix C: Example of a COM toolbar plugin (on page 27)

The fourth appendix gives details of the Maxthon plugin commands.

Appendix D: Maxthon plugin commands (on page 31)

1.4 Acronyms and abbreviations

Acronyms and glossary terms that appear in Maxthon documentation include:

Table 1: Acronyms and Glossary Terms

Abbreviation	Description
COM	**Component Object Model** - a Microsoft platform for software componentry introduced by Microsoft in 1993. COM is used to enable interprocess communication and dynamic object creation in any programming language that supports the technology.
CSS	**Cascading Style Sheets** - style sheets comprise external style rules that tell a browser how to present a document.
DHTML	**Dynamic HTML** - a term used for a collection of technologies, used together to create interactive web sites by using a combination of a static markup language (such as HTML), a client-side scripting language (such as JavaScript), a presentation definition language (Cascading Style Sheets), and the Document Object Model.
DOM	**Document Object Model** - the tree structure of HTML documents. DOM provides an object oriented interface that allows parsing HTML into a well defined tree structure and operating on its contents. Maxthon plugins can then manipulate contents of HTML pages based on this defined structure.
HTML	**HyperText Markup Language** - a language designed for the creation of web pages with hypertext and other information to be displayed in a web browser.

2. Understanding Maxthon plugins

In this chapter

2.1 What is a Maxthon plugin?

Maxthon plugins add functionality to the Maxthon Internet Browser. A plugin aims to enhance the browsing experience by displaying a new page in the browser, affecting something on an existing page, or launching an external tool.

Typically, plugins are used for the following purposes:

- To integrate an existing web service (such as a search engine, email service, or blog site) with the browser
- To integrate an existing local application with the browser
- To manipulate the display, elements, and features of a page already displayed in the browser

2.1.1 Examples of existing plugins

An enormous variety of Maxthon plugins have been created. To get an idea of the range of functionality available with Maxthon plugins, see the sample below.

Table 2: Sample list of plugins available for Maxthon 1.5.x

Plugin Name	Description
GoogleSearch	*GoogleSearch* is a bundle of seventeen search plugins accessible from a single button on the Maxthon toolbar. These scripts provide access to the following Google searches: Web, Images, Groups, News, Blog, Froogle, Directory, Microsoft, Glossary, Maps, Print, Video, I'm Feeling Lucky, and to site specific services: Site Search, Similar to Current, Linking to current, and Cached Page.
Remove It	*Remove It* allows users to remove any specific contents from web pages, for example tables of text adverts, banner adverts, and so on.
Flickr Sidebar	*Flickr Sidebar* provides immediate access to photos from Flickr.com (a very popular online photo management and sharing application) in the Maxthon Sidebar. The plugin allows users to search and upload photos as well as view their own photos, and those recently added by other Flickr users.
Del.icio.us Sidebar	*Del.icio.us Sidebar* loads favorites from the del.icio.us service (social bookmarking service for storing and sharing web bookmarks) into the Maxthon Sidebar.

Plugin Name	Description
Technorati	*Technorati* provides access to Technorati (search engine that indexes over 11 million blogs) searches in a variety of ways. The plugin can search Technorati for: • a selected word or phrase on the open page • blogs that comment on or link to the open page The plugin also provides shortcuts to Technorati personalized favorites and the "popular", "discover", and "watchlist" tags.
MSN Web Messenger	*MSN Web Messenger* loads the official MSN Web Messenger into the Maxthon Sidebar. Chat windows show up as pop-up windows in the main window.
Gmail Notifier	*Gmail Notifier* checks multiple Gmail accounts at user defined intervals and provides message box alerts for new mail.
EnableRightClick	*EnableRightClick* enables the right click menu on sites that block it.

Note: All of these plugins are available for download from the *Maxthon Resource Center* http://res.maxthon.com/ItemList.aspx?t=2].

2.2 Why create a Maxthon plugin?

Five typical reasons why people have created Maxthon plugins:

- To increase the potential number of users for your web service by providing support for an additional browser
- To reduce time spent on performing repetitive tasks
- To quickly pass information to local applications
- To control local applications from within the browser
- To change an existing web site in appearance or function

2.2.1 To add Maxthon support to your service

Organizations that provide web services aim for as much cross-browser support as possible in order to generate more interest and Internet traffic. These sites often have existing plugins for Internet Explorer and it is a minimal amount of work to produce a Maxthon plugin. Since Maxthon uses the IE engine, some IE plugins are compatible with no effort at all.

2.2.2 To reduce time spent on repetitive tasks

Using a plugin, web surfers may be able to save time by automating repetitive browser-based tasks. For example, when reading a movie review you might want to perform an IMDB search for the movie.

In IE you would:

1. Select the title
2. Copy the title
3. Open IMDB.com
4. Paste the title into the search box
5. Click search

The Maxthon *IMDB plugin* reduces this to:

1. Select the title
2. Click the plugin button to open a new tab with the IMDB search results

An obvious use for this type of plugin is for bloggers who wish to reduce the time it takes to create blog entries. Blogger friendly plugins such as *BlogDrive* and *BlogLines* allow users to highlight text on a page and quickly create blog posts.

2.2.3 To pass information to local applications

Web surfers who need to provide details from the web to specific local applications can take advantage of the powers of Maxthon plugins and allow a plugin to handle this chore. For example, a plugin could be created to simplify the process of sending emails containing some highlighted text together with the URL of the page from a local mail application (such as Microsoft Outlook or Mozilla Thunderbird).

2.2.4 To control local applications from within the browser

Some plugins have been created in order to bring as many frequently used applications as possible inside the browser.

Plugins which allow control of local applications include plugins that control Winamp, Windows Media Player, MSN Messenger, and ICQ.

2.2.5 To change the appearance and functionality of a page

Some pages would benefit from a redesign. Whether it is changing the background color or removing annoying adverts, or even adding entirely new features, Maxthon plugins can help.

Maxthon plugins are capable of fundamentally changing the way a site is displayed and making the user experience more pleasant or more useful.

When Gmail was first launched it did not have HTML support or a **Delete** button. Maxthon plugins soon appeared solving both of those issues. Maxthon plugin developers were, in effect, providing a superior Gmail user experience to Maxthon users.

2.3 What are the differences between Maxthon and IE?

Developers considering creating a browser plugin will find more power and functionality in Maxthon than IE.

Three reasons to make a Maxthon plugin rather than just writing a plugin for IE:

- Maxthon's tabbed interface allows greater flexibility when opening new pages from your plugin.
- Maxthon's built-in Plugin Commands (providing extended DHTML support) offer access to features not found in IE. For more details, see *Appendix D: Maxthon plugin commands* (on page 31).
- Maxthon's plugins are stored in a location that is logical for end users. This is increasingly important as more people carry their applications in portable formats.

Below are three points to be aware of when creating a Maxthon plugin:

- Maxthon is a tabbed browser. This means that more than one WebBrowser control is running and toolbars must be capable of working with more than one web page.
- Maxthon passes the IWebBrowser2 pointer of the active tabs to toolbars. When a tab is activated, Maxthon will call a toolbar's IObjectWithSite::SetSite command and pass the IWebBrowser2 pointer of the tab to toolbar. The toolbar must release the previous pointer and use the new one, in order to work on the new page.
- Some IE properties may not work in Maxthon. These are usually properties such as `get_hwnd`, which MSDN acknowledge will not work in applications that host multiple WebBrowser controls. Maxthon is evolving all the time and when reasonable requests for additional properties are presented, the development team will investigate the possibility of implementing them natively.

3. Types of Maxthon plugin

In this chapter

3.1 Script, COM and EXE

Three types of plugin exist for Maxthon as detailed below. To decide which type of plugin to produce, consider the functionality required and the expertise of your software team. Also, if you have an existing plugin then it may be simpler to simply adapt that. For more information, see *Create a new plugin or adapt an existing one?* on page 13.

The three types of Maxthon plugin:

- **Script plugins** can be written in any valid programming language supported by IE, such as JavaScript. Script plugins are accessed in Maxthon from a button in either the Plugins Bar or the Sidebar.

  When a user clicks on the button for the plugin, the script is run. In the case of script sidebar plugins, this usually leads to an HTML page opening in the Sidebar similar to the page shown in the Sidebar of Figure 1: Respective locations of the Sidebar and Plugins Bar (see page 12).

  The Maxthon community mainly writes JavaScript plugins, since these are easiest to produce for users with minimal programming experience.

- **COM plugins** are typically for more experienced developers than script plugins. They are accessed as a complete toolbar, a toolbar button, or a sidebar. Most Internet Explorer plugins are COM based and adapting an existing IE plugin for Maxthon compatibility is usually a small amount of work.

  Once your plugin works in Maxthon, you may decide to extend the feature-set of the plugin in order to use some of the Maxthon-specific capabilities. At this stage, it might prove beneficial to split the development into two separate plugins: one for IE and the other for Maxthon.

- **EXE plugins** are local executable files which Maxthon launches with special command line parameters such as the URL of the current page, the handle of the current Maxthon window, and so on. These features are also available in the *external utilities* feature of Maxthon. EXE plugins differ as they are typically applications that are designed to work with Maxthon. Also, applications launched as *external utilities* have to be manually configured and cannot be installed as plugins.

3.2 Sidebar, button, and toolbar

Sidebar plugins are used for plugins that are intended to display information to the user. For example weather, favorites, calendar, pictures, and so on. They are usually unrelated to the current page/s being shown in the main window of the browser and that is why they can be shown in the separate pane without confusion.

Button plugins tend to be designed to interact with the current page being viewed. Button plugins might save all images on the open page, verify the URL is genuine (as an anti-phishing precaution) and so on. Button plugins are also used to launch new tabs related to the current page or some of the content on that page, for example to run a custom search at Amazon for some selected text.

Figure 1: Respective locations of the Sidebar and Plugins Bar

Script plugins can be run from the Plugins Bar or the Sidebar.

COM plugins can be run from the Sidebar or as complete toolbars. "Complete Toolbars" are groups of related programs that offer more as a group than they would separately. Examples include the *Google Toolbar* that has many Google related items in a single toolbar to allow users to access all of these different services from one unified toolbar.

EXE plugins can be run from the Plugins Bar.

4. Developing a Maxthon plugin from an existing IE plugin

In this chapter

4.1 Create a new plugin or adapt an existing one?

When deciding whether to create a new plugin or adapt an existing one, you should first test your existing plugin for Maxthon compatibility. It is possible that the plugin already works with the browser and that no additional programming is needed.

The rest of this chapter explains how to install and access your plugin.

If you find that the plugin works with Maxthon, explore Maxthon's considerably more powerful plugin support and decide whether there are possibilities for extending your plugin beyond the more limited IE framework.

4.2 Core elements of every Maxthon plugin

Every Maxthon plugin must have the following three files present:

- An INI file - always called **plugin.ini** - according to the framework set out in *The plugin.ini file* on page 14.
- An icon file to use for the Sidebar or button - this should be a valid ICO file which is specified in the INI file
- An HTML/DLL/EXE (for script, COM, or EXE plugins respectively) file containing the script to run when the icon is clicked

4.3 The plugin.ini file

An initialization file must be included in every plugin folder. This file, plugin.ini, contains the details of the plugin as described in the following table.

Table 3: Fields of the plugin.ini file

Field	Description
[General]	Standard header that must be present in every plugin.ini.
Name=	Name of the plugin.
Author=	Author of the plugin.
Version=	The revision number of the plugin.
ModuleType=	The type of plugin. Options are COM, EXE, and SCRIPT. For more information, see *Script, COM and EXE* (on page 11).
FileName=	The name of the HTML/DLL/EXE (for script, COM, and EXE plugins respectively) file to run when the button is clicked.
Comments=	These comments will be shown to end users via the Maxthon Plugin options window.
Type=	The script type. Options are M2Plugin_BUTTON, M2Plugin_SIDEBAR, or M2Plugin_Toolbar.
HotIcon=	Icon to use when a user moves their mouse over the plugin button.
Icon=	Icon to use whenever the user's mouse is not hovering over the plugin button. This can be the same as HotIcon.
[Settings]	Optional settings area for adding additional parameters.

4.4 Installing a Maxthon plugin

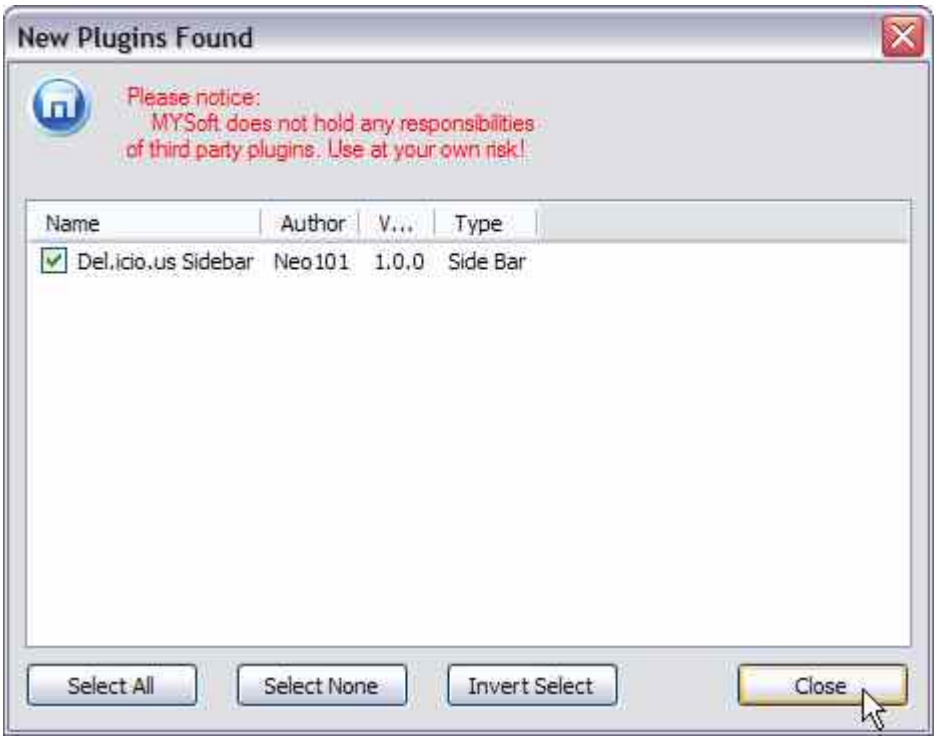
Once you have written your plugin.ini file, created your script file (HTML, DLL, or EXE), and prepared at least one icon, you can install the plugin into Maxthon.

To install a Maxthon plugin:

1. Close Maxthon if it is running.
2. Open the Maxthon Plugin folder.

   The default location is C:\Program Files\Maxthon\Plugin.
3. Create a new folder with the name of your plugin.
4. Copy the files needed for your plugin into the new folder.
5. Run Maxthon.

   A window will appear asking you to confirm the installation of the newly found plugin.



6. Ensure the check box for your plugin is selected, and click **Close**.

   Your plugin should now appear in Maxthon.

4.5 Using button plugins

To use a button plugin:

1. Turn on the Plugins Bar:
   a) From the View Menu, select Toolbars.
   b) Select Plugins Bar.

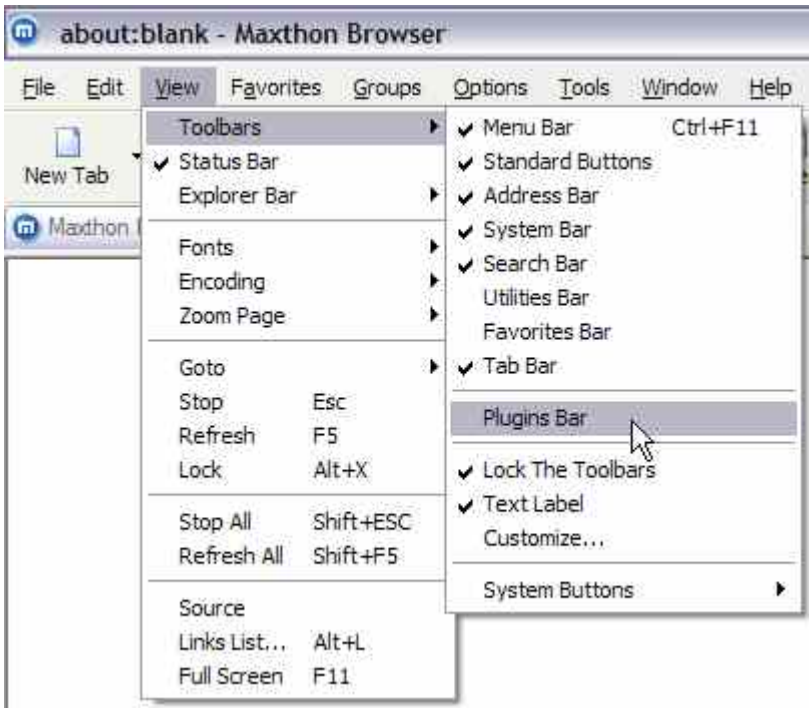


Figure 2: Turning on the Plugins Bar

A new toolbar should now be shown in the toolbar area of your Maxthon window as shown in the following figure.

Figure 3: An example Plugins Bar with thirteen plugins

2. Click the relevant button for the desired plugin.

Note: Four of the plugins shown in the previous figure have small dropdown arrows to the right of the icon. These arrows represent multiple plugins combined into a single package.

For example, the *GmailThis* plugin has two uses: sending a page via Gmail and opening a blank Gmail to compose a new message. Clicking the dropdown arrow alongside the plugin icon opens the menu as shown in the following figure:

4.6 Using sidebar plugins

To use a sidebar plugin:

1. Open the Sidebar:

   a) From the Standard Buttons toolbar, click the Side Bar button 

; or press CTRL+I.

The Sidebar is now shown.

Figure 4: Sample Sidebar with eight plugins below the six standard Sidebar features

2. If you do not wish to use the Sidebar but want access to a sidebar plugin, you can access them all from the View Menu:

   a) From the View Menu select Explorer Bar.

      A list of available sidebar plugins is shown as shown in the following figure.

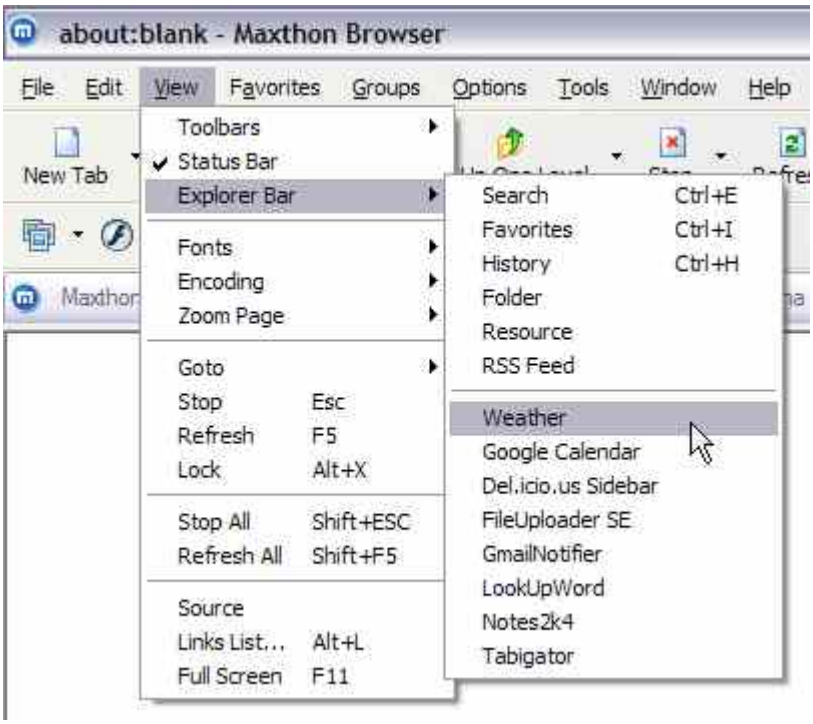


Figure 5: Select a sidebar plugin from the View Menu

4.7 Troubleshooting

The most common technical support query regarding IE plugins not working in Maxthon is that the toolbar or sidebar buttons simply do not appear.

Typically, plugins which were not designed for Maxthon have errors because of one or both of the following:

- The plugin uses specific IE properties which are not supported by Maxthon
- The plugin is designed to work with only one browser window and Maxthon's multiple tab control environment was not considered in the design stage

Maxthon is happy to work with developers to resolve these issues.

Contact Maxthon (Asia) Ltd. technical support at: develop@maxthon.com.

4.8 Further reading

For additional material related to Maxthon plugins, see the following pages:

- Online tutorial for creating script plugins:
  http://maxthon.neo101.nl/tutorial
- Maxthon Browser Scripting Reference:
  http://beta.maxthon.com/doc/dev/script.htm
- Maxthon plugin and resources forum:
  http://forum.maxthon.com/?showforum=38

Appendix A: Example of a script button plugin

The plugin outlined in this appendix adds a button to your Plugins Bar. This button runs a script that examines the current page and then launches a new tab showing all images from the current page.

As explained in *Core elements of every Maxthon plugin* (on page 13), the essential components of a script plugin are:

- A plugin.ini file with the details of the plugin (see *The plugin.ini file* on page 14)
- An icon file (*.ico) for the Plugins Bar
- An HTML file containing a script to run or a page to be displayed

Sample plugin.ini file

The structure of the plugin.ini file is the same regardless of the type of plugin. This file instructs the browser about the type and version of the plugin as well as other key details.

```
[General]
Name=Image Extractor Toolbar Example
Author=Neo101
Version=1.0.0
ModuleType=SCRIPT
FileName=imageExtractor.html
Comments=Toolbar example plugin
Type=M2Plugin_BUTTON
HotIcon=image.ico
Icon=image.ico
[MyIE2Buttons]
Count=1
Name1=Configuration...
FileName1=Config_caller.html
```

Note: There is an additional section of the plugin.ini which was not shown in *The plugin.ini file* on page 14. The [MyIE2Buttons] section enables you to use a single icon with a dropdown list to access multiple plugin scripts. For more information, see the note at the end of *Using button plugins* on page 15.

The plugin.ini above instructs Maxthon to use the icon named image.ico as shown in the following figure.

Figure 1: The Plugins Bar button for the plugin

This plugin takes information from the current tab. If you have the page *Maxthon.com* open when you click the icon shown above, the page shown in the following figure will open.

Figure 2: The HTML page

The page shown is imageExtractor.html as dictated by the plugin.ini file. The plugin has taken the images from the page that was being browsed (maxthon.com) and is showing only the images from that page together with the URLs of those images and some additional options (save, open, add to adhunter).

The original HTML file is shown below to show how the above page was generated.

The HTML code

Note: The HTML code is actually JavaScript enclosed in HTML tags and is not 'pure' HTML.

```
<script language="javascript">
(function(){
//The security id you need for special Maxthon commands
var SECURITY_ID = %max_security_id
//This name should match that in plugin.ini
var PLUGIN_NAME = 'Image Extractor Toolbar Example'
//The folder where the plugin is located
var PLUGIN_FOLDER = external.m2_plugin_folder(SECURITY_ID, PLUGIN_NAME)
//The file where the settings are stored
var SETTINGS_FILE = 'settings.ini'
```

```
//The absolute locations of the stylesheet and script
var stylesheetLocation = PLUGIN_FOLDER + 'stylesheet.css'
var scriptLocation = PLUGIN_FOLDER + 'imageExtractor.js'
//Reads a setting from the settings file
var filterSmallImages = readSetting('filterSmallImages', '1')
//imagesArray contains all images found on the current page
var imagesArray = returnAllImages()

//page HTML is written to variable 'html'
var html = returnWebpageHead()
html += returnWebpageImages( imagesArray )
html += returnWebpageBottom()

//Open a new blank window and write the HTML to this new window
var win = openWindow()
win.document.write(html)

//Set some variables on the new window that we might need later on
win.SECURITY_ID = SECURITY_ID
win.PLUGIN_NAME = PLUGIN_NAME

//Make sure you can't add new HTML to the window
win.document.close()

function returnAllImages(){
 return document.getElementsByTagName('img')
}

//Returns the head of the web page
function returnWebpageHead(){
 var html = '<html>'+
                        '<head>'+
                                '<title>Images Extractor</title>'+
                                '<link href="' + stylesheetLocation + '"
type="text/css" rel="stylesheet">'+
                                '<script type="text/javascript" src="' +
scriptLocation + '"><\/script>'+
                        '</head>'+
                        '<body>'+
                                '<h1>Image Extractor</h1>'+
                                '<div class=from>Images from ' + location.href +
'</div>'
 return html
}

//Most important part of the script - puts the images inside the HTML code
function returnWebpageImages( array ){
 var html = '<div class=images>'

 //All URLs from the images will be stored in the Array. To avoid duplicate URLs
 var writtenImages = new Array()

 //iterate over all images
 for(var i=0; i<array.length; i++){
        var imageObject = array[i]

        //Check if the image already written to HTML source. (avoids duplicates)
        if( inArray( imageObject.src , writtenImages ) ){
                continue
        }

        //Checks if enabled the small image filter enabled. If so, it checks if
the image is smaller that 50px on any side
        if( filterSmallImages == '1' && ( imageObject.height < 50 ||
imageObject.width < 50 ) ){
                continue
        }

        //Add the URL to the writtenImages array
        writtenImages.push( imageObject.src )

        //Write the HTML code
        html += '<div class=imageDiv>' +
```

```
                                    `<img src="` + imageObject.src + `" alt="` +
imageObject.src + `"><br>`+
                                    `<div class=options>`+
                                           `<span onclick=save("` + imageObject.src +
`") class=option>Save</span> | `+
                                           `<span onclick=addToAdhunter("` +
imageObject.src + `") class=option>Add to Adhunter</span> | `+
                                           `<span onclick=openInNewWindow("` +
imageObject.src + `") class=option>Open</span>` +
                                    `</div>`+
                                    `<input type=text value="` + imageObject.src + `"
onclick="this.select()" class=urlInput><br>`+
                             `</div>`
 }
 //Show an error message when no images are found
 if( writtenImages.length == 0){
        html += `<div class=message>No images found!</div>`
 }
 html += `</div>`
 return html
}

function returnWebpageBottom(){
 return `</body></html>`
}

//Checks if <text> found as member of an array. Returns true if so, else false.
function inArray( text, array ){
 for(var i=0; i<array.length; i++){
        if(text == array[i]){
               return true
        }
 }
 return false
}

//reads a Key from the settings file and returns it
function readSetting(Key, errorResponse) {
 if(errorResponse == null){
        errorResponse = ``
 }
 var returnValue = external.m2_readIni(SECURITY_ID, PLUGIN_NAME, SETTINGS_FILE,
"Settings", Key, errorResponse)
 if(returnValue == undefined){
        return errorResponse
 } else {
        return returnValue
 }
}

//Open a new window and return the HTML DOM object for this tab
//This function is more compex than usual: win = window.open(`about:blank`)
//because that syntax causes "access denied" error on sites like cnn.com
function openWindow(){
 //this will focus the new window & ensure Ad hunter doesn't block this window
 external.m2_callerName(SECURITY_ID, PLUGIN_NAME)

 //Opens the new window
 window.open(`about:blank`, `_blank`)

 //Returns the tab DOM
 return external.get_tab(SECURITY_ID, external.cur_sel)
}
}())
</script>
```

Note: This example is of a *basic* plugin. There are many things you might like to add, such as support for BB-Code or HTML links, more settings, frames support, options for resizing large images, display of "alt" or "title" from image, etc.

Appendix B: Example of a script sidebar plugin

The plugin outlined in this appendix adds a button to your Sidebar to facilitate a quick Yahoo search.

As explained in *Core elements of every Maxthon plugin* (on page 13), the essential components of a script plugin are:

- A plugin.ini file with the details of the plugin (see *The plugin.ini file* on page 14)
- An icon file (*.ico) for the Plugins Bar
- An HTML file containing a script to run or a page to be displayed

Sample plugin.ini file

The structure of the plugin.ini file is the same regardless of the type of plugin. This file instructs the browser about the type and version of the plugin as well as other key details.

```
[General]
Name=Yahoo Search Sidebar Example
Author=Neo101
Version=1.0.0
ModuleType=SCRIPT
FileName=yahoo.html
Comments=Example plugin
Type=M2Plugin_Sidebar
HotIcon=yahoo.ico
Icon=yahoo.ico
```

The plugin.ini instructs Maxthon that the icon to use is yahoo.ico as shown in the following figure.

Figure 1: The Sidebar button for the plugin

Clicking the icon in the Sidebar will launch the page shown in the following figure.

Figure 2: The HTML page

The page shown is yahoo.html as dictated by the plugin.ini file. The HTML file is shown below to show how the above page was generated.

The HTML code

```
<html>
<head>
  <link href="stylesheets/StyleSheet.css" type="text/css" rel="stylesheet">
  <script type="text/javascript" src="max.src"></script>
  <script type="text/javascript" src="scripts/Yahoo.js"></script>
  <script type="text/javascript" src="" id="searchScript" charset="utf-8"></script>
</head>
<body>
  <h1>Yahoo Search</h1>

  <input type=text id=keyword>
  <input type=submit onclick="submit()" value="Search">

  <div id=numberOfResults></div>

  <div class=nextPrevious>
         <span id=previousTop></span>
         <span id=nextTop></span></div>

  <div id=results>      </div>

  <div class=nextPrevious>
         <span id=previous></span>
         <span id=next></span></div>

  <div class=about>
         Plugin created by <a href="http://www.neo101.nl/" target=_blank
title="www.neo101.nl">Neo101</a><br>
         Version 1.0.0<br>
         <!-- Begin Yahoo Web Services HTML Attribution Snippet -->
         <a href="http://developer.yahoo.net/about/" target=_blank>
                Web Services by Yahoo!</a>
         <!-- End Yahoo Web Services HTML Attribution Snippet --></div>
</body>
```

Appendix C: Example of a COM toolbar plugin

When the code shown below is compiled into a DLL it becomes a simple COM toolbar plugin. It's main purpose is as a reference for the method of creating a COM toolbar plugin. The code is available in the Maxthon SDK which can be downloaded from http://www.maxthon.com/files/sdk.zip.

```
// HelloBar.h : Declaration of the CHelloBar
#ifndef __HELLOBAR_H_
#define __HELLOBAR_H_

#include "resource.h"       // main symbols
#include <shlobj.h>
#include <docobj.h>
#include <ExDisp.h>
#include "mshtml.h"
#include <atlwin.h>
#include <ExDispid.h>
#include "imyie.h" //include Maxthon interface

/////////////////////////////////////////////////////////////////////////////
// CHelloBar
class ATL_NO_VTABLE CHelloBar :
  public CComObjectRootEx<CComSingleThreadModel>,
  public CComCoClass<CHelloBar, &CLSID_HelloBar>,
  public IDispatchImpl<IHelloBar, &IID_IHelloBar, &LIBID_HELLOTOOLBARLib>,
  public IObjectWithSite,
  public IDeskBand,       //IDeskband interface
  public IMyIEClient
{
public:
  CHelloBar()
  {
  }

DECLARE_REGISTRY_RESOURCEID(IDR_HELLOBAR)
DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CHelloBar)
  COM_INTERFACE_ENTRY(IDispatch)
  COM_INTERFACE_ENTRY(IMyIEClient)
  COM_INTERFACE_ENTRY(IDeskBand)
  COM_INTERFACE_ENTRY(IOleWindow)
  COM_INTERFACE_ENTRY(IObjectWithSite)
END_COM_MAP()

// IHelloBar
//------------IObjectWithSite---------------------//
  virtual HRESULT STDMETHODCALLTYPE SetSite(IUnknown __RPC_FAR *pUnkSite)
  { // this function is called when the tab is created or activated
          if(m_pWeb)      //already connected? disconnect it.
          {
                  CComQIPtr<IConnectionPointContainer> pConn = m_pWeb;
                  if(pConn)
                  {
                          CComPtr<IConnectionPoint> spCP;
                          if(SUCCEEDED(pConn-
>FindConnectionPoint(DIID_DWebBrowserEvents2, &spCP)))
                                  spCP->Unadvise(m_cookie);
                  }
                  m_pWeb.Release();
          }
          if(m_Label.m_hWnd==NULL) //create the static
          {       //create label

                  //trying to get parent window
```

```
                HWND hParent = NULL;
                CComQIPtr<IOleWindow> pWin=pUnkSite;
                if(pWin)
                        pWin->GetWindow(&hParent); //Get parent window of the
toolbar
                if(hParent)
                {
                        m_hwndMaxthon = hParent;
                        RECT r={0,0,100,30};
                        m_Label.Create( "Static", hParent, r, "Hello
World",WS_CHILD ); //create the window
                }
        }

        m_pWeb = pUnkSite; //save current tab's IWebBrowser2 pointer
        CComQIPtr<IConnectionPointContainer> pConn = m_pWeb;   //connect to web
events
        if(pConn)
        {
                CComPtr<IConnectionPoint> spCP;
                if(SUCCEEDED(pConn->FindConnectionPoint(DIID_DWebBrowserEvents2,
&spCP)))
                        spCP-
>Advise(reinterpret_cast<IDispatch*>(this),&m_cookie);
        }
        ShowRealDomain(); //refresh text
        return S_OK;
  }
STDMETHODIMP GetIDsOfNames(/* [in] */ REFIID riid,
                                                /* [size_is][in] */ OLECHAR**
rgszNames,
                                                /* [in] */ UINT cNames,
                                                /* [in] */ LCID lcid,
                                                /* [size_is][out] */ DISPID*
rgDispId)
{
  return S_FALSE;
}
//where the event goes
STDMETHODIMP Invoke(DISPID      dispidMember,
                                                   REFIID      riid,
                                                   LCID        lcid,
                                                   WORD        wFlags,
                                                   DISPPARAMS* pDispParams,
                                                   VARIANT*    pvarResult,
                                                   EXCEPINFO*  pExcepInfo,
                                                   UINT*       puArgErr)
{
  switch(dispidMember) {
  case DISPID_BEFORENAVIGATE2:
        {
                m_bstrDomain = pDispParams->rgvarg[5].pvarVal->bstrVal;
                TCHAR* pStr = NULL;
#ifdef UNICODE
                pStr = &m_bstrDomain;
#else
                USES_CONVERSION;
                pStr = W2A(m_bstrDomain);
#endif
                m_Label.SetWindowText(pStr);
                break;
        }
  case DISPID_DOCUMENTCOMPLETE:
        {
                ShowRealDomain();
                break;
        }
  }
  return S_OK;
}
    virtual HRESULT STDMETHODCALLTYPE GetSite(
            /* [in] */ REFIID riid,
            /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvSite)
  {
```

```
        return S_FALSE;
  }

  //-----IOleWindow------------------//
  //return my window to Maxthon
  STDMETHOD(GetWindow) (THIS_ HWND * lphwnd)
  {
        *lphwnd = m_Label.m_hWnd;
        return S_OK;
  }
    STDMETHOD(ContextSensitiveHelp) (THIS_ BOOL fEnterMode)
  {
     return S_OK;
  }

    // *** IDockingWindow methods ***
  // Show my window
    STDMETHOD(ShowDW)         (THIS_ BOOL fShow)
  {
        m_Label.ShowWindow(fShow?SW_SHOW:SW_HIDE);
        return S_OK;
  }
  // Destroy my window
    STDMETHOD(CloseDW)        (THIS_ DWORD dwReserved)
  {
        m_Label.DestroyWindow();
        return S_OK;
  }
    STDMETHOD(ResizeBorderDW) (THIS_ LPCRECT   prcBorder,
                                     IUnknown* punkToolbarSite,
                                     BOOL      fReserved)
  {
     return E_NOTIMPL;
  }

//-----------------IDeskBand-------------------------------------------//
  // return size of my window, so Maxthon knows.
  STDMETHOD(GetBandInfo) (DWORD dwBandID, DWORD dwViewMode,
                                DESKBANDINFO* pdbi)
  {
        RECT r;
        m_Label.GetWindowRect(&r);
        pdbi->ptMinSize.x = r.right - r.left;
        pdbi->ptMinSize.y = r.bottom- r.top;
        return S_OK;
  }

//-----------------IMyIEClient-----------------------------------------//
  //Maxthon asks for initialize
  BOOL Init(IMyIEServer* pServer)
  {
        m_pMyIEServer = pServer;
        pServer->AddRef();
        HWND hwnd = pServer->GetInfo()->hMainWnd;
        return TRUE;
  }
  //Maxthon exits
  BOOL Exit()
  {
        m_pMyIEServer->Release();
        return TRUE;
  }
  //Maxthon asks to show the config window of this plugin
  BOOL Config(HWND hParent)
  {
        MessageBox(hParent,"No configration so far","Thanks",MB_OK);
        return TRUE;
  }
public:
  void ShowRealDomain()
  {
        if(m_pWeb)
        {
              CComPtr<IDispatch> pDisp;
```

```
                m_pWeb->get_Document(&pDisp);
                CComQIPtr<IHTMLDocument2> pDoc = pDisp;
                if(pDoc)
                {
                        pDoc->get_domain(&m_bstrDomain);
                        SetupText();
                }
        }
  }
  void SetupText()
  {
        TCHAR strToDisplay[1024] = _T("You are on site:  ");
#ifdef UNICODE
        _tcscat(strToDisplay,m_bstrDomain);   //need length check
#else
        USES_CONVERSION;
        char* pStr = W2A(m_bstrDomain);
        if(pStr)
                _tcscat(strToDisplay,pStr);
#endif
        m_Label.SetWindowText(strToDisplay);
  }
protected:
  IMyIEServer* m_pMyIEServer; //Hold Maxthon Server pointer
  CComQIPtr<IWebBrowser2> m_pWeb;
//        CAxWindow m_HTML;
  CWindow m_Label;
  HWND m_hwndMaxthon;
  DWORD m_cookie;
  CComBSTR m_bstrDomain;
};
#endif //__HELLOBAR_H_
```

Appendix D: Maxthon plugin commands

Maxthon extends the DHTML support of IE by adding several new commands which can be called by any scripting language such as Javascript and VBScript.

These APIs can be called via the window.external object in an HTML page or from a Maxthon script plugin.

Some of the functions below, require `security_id`. This is a variable that looks like this: "{DCFC4598-B97C-49B6-8108-EE86F8694283}". The security_id changes every time Maxthon is restarted.

To get a security_id for a plugin:

1. For button plugins, assign the variable %max_security_id to security_id in your script, as follows:

   ```
   var security_id=%max_security_id;
   ```

   The variable `security_id` can now be used in your script.

2. For sidebar plugins, a file named max.src will be created in your plugin's folder. Include this file in your HTML file using the following command:

   ```
   <script type="text/javascript" src="max.src"></script>
   ```

   The variable max_security_id can now be used in your script in place of `security_id`.

Table 1: Maxthon plugin commands

Command	Description
Extended Properties	
max_version	Get current version of Maxthon Browser
max_language_id	Get current language id of Maxthon UI
current_tab (cur_sel)	Get current tab's index
tab_count	Get current opened tab's count
General Methods	
addFavorite()	Call the Add Favorites dialog
addProxy()	Call the Add Proxy dialog
addFilter()	Call the Add Filter dialog
max_addProxyProvider()	Call the Add Proxy Provider dialog
m2_run_cmd()	Execute Maxthon functions by Command ID
m2_search_text()	Get search string in Maxthon's Search Box
m2_plugin_folder()	Get local folder path of a plugin
Tab Related Methods	
get_tab()	Get a tab's window object by index

Command	Description
activate_tab()	Activate a tab by index
close_tab()	Close a tab by index
File Related Methods	
readFile()	Read data from a file
writeFile()	Write data to a file
m2_readIni()	Read an INI file
m2_writeIni()	Write to an INI file

Note: A web-based reference for Maxthon plugin commands (including usage examples) can be found here: http://beta.maxthon.com/doc/dev/script.htm.

Index