mDolphin Plug-in
Programming Guide

Version 2.0
For mDolphin Version 2.0

Beijing Feynman Software Technology Co. Ltd.
March, 2008
Contents

1 About This Guide ................................................................. 1
   1.1 Scope ........................................................................ 1
   1.2 Audience .................................................................. 1

2 About Plug-ins .................................................................... 2
   2.1 HTML Tags Used to Display Plug-ins ......................... 2
      2.1.1 About the object tag .......................................... 2
      2.1.2 About the embed tag ......................................... 3

3 Developing Plug-ins .......................................................... 5
   3.1 Identifying a Plug-in .................................................. 5
   3.2 Register and UnRegister Plug-ins ............................... 5
      3.2.1 Register a Plug-in to mDolphin ......................... 5
      3.2.2 UnRegister a Plug-in to mDolphin ..................... 5
   3.3 Initializing and Destroying Plug-ins ............................ 6
      3.3.1 Plug-in initialization .......................................... 6
      3.3.2 Creating a plug-in instance ............................... 7
      3.3.3 Destroying a plug-in instance ........................... 8
      3.3.4 Shutdown .......................................................... 9
   3.4 Drawing Plug-ins ....................................................... 10
   3.5 Allocating and Freeing Memory .................................... 10
   3.6 Implementing Streams ............................................... 11
      3.6.1 Sending a stream from the browser to a Plug-in .... 11
   3.7 Handling URLs ....................................................... 12
      3.7.1 Retrieving data from a URL ............................... 12
      3.7.2 Posting URLs ................................................ 12

4 Plug-in API Reference Tables ............................................. 14
   4.1 Adapted Netscape Plug-in API Functions .................... 14
      4.1.1 Initialization and destruction functions .................. 14
      4.1.2 Drawing functions .......................................... 16
      4.1.3 Stream functions ............................................ 18
      4.1.4 URL functions ............................................... 21
      4.1.5 Memory functions ........................................... 24
      4.1.6 Utility functions ............................................. 25
4.1.7 Java communication functions .................................................. 26

4.2 Extensions .............................................................................. 26
  4.2.1 mdolphin_register_plugin...................................................... 26
  4.2.2 mdolphin_unregister_plugin................................................... 26
  4.2.3 mdolphin_get_plugin_counts.................................................. 27
  4.2.4 mdolphin_get_plugin_from_mimetype..................................... 27
  4.2.5 mdolphin_get_plugin_info..................................................... 27
  4.2.6 mdolphin_get_plugin_info_by_index..................................... 28

4.3 Structures ................................................................................ 28
  4.3.1 NPByteRange ..................................................................... 28
  4.3.2 NPEmbedPrint ..................................................................... 28
  4.3.3 NPFullPrint ......................................................................... 28
  4.3.4 NPP .................................................................................... 28
  4.3.5 NPPrint .............................................................................. 29
  4.3.6 NPRect .............................................................................. 29
  4.3.7 NPSavedData ...................................................................... 29
  4.3.8 NPStream .......................................................................... 29
  4.3.9 NPWindow .......................................................................... 31

5 Hello World Plug-in ...................................................................... 33
  5.1 Default Plug-in Demo ............................................................... 33
  5.2 Implementing Hello World Plug-in .......................................... 33
    5.2.1 Modifing Project Name ......................................................... 33
    5.2.2 Define Plug-in Name and MIME Type .................................. 33
    5.2.3 Implementing Hello World Plug-in ..................................... 34
  5.3 Building and Installing Plug-in ................................................. 35
    5.3.1 Building Hello World Plug-in ............................................ 35
    5.3.2 Installing Plug-in ............................................................... 35
  5.4 Write Test Html for Hello World Plug-in ................................... 36
1 About This Guide

This guide explains how to develop plug-ins for mDolphin, using the Browser Plug-in API. The Browser Plug-in API was introduced in chapter 4.

1.1 Scope

This document contains the following information:

- Chapter 1 provides an introduction to this document, including its scope, intended audience.
- Chapter 2 contains basic information about plug-ins.
- Chapter 3 provides instructions for developing a plug-in application.
- Chapter 4 describes the functions and structures of the Browser Plug-in API, grouped by functionality.
- Chapter 5 describes how to implement a hello world plug-in.

1.2 Audience

This guide is intended for developers who wish to write plug-in applications for the mDolphin. The reader should be familiar with the MiniGUI programming and the linux operating system.
2 About Plug-ins

A plug-in is an add-on program that extends the capabilities of a browser. For example, plug-ins can enable users to view pdf files and flash files, or to play audiotapes or movies on a browser. The Browser Plug-in API enables developers to create plug-ins that can do the following:

- Register one or more MIME types.
- Draw inside a browser window.
- Receive key and mouse events.
- Obtain data from the network using URIs.
- Post data to URIs.
- Communicate with Javascript from native code.

2.1 HTML Tags Used to Display Plug-ins

HTML tags determine the way a plug-in is displayed on a Web page. The following HTML tags invoke the plug-in and determine its display mode:

- object.
- embed.

2.1.1 About the object tag

The object tag specifies the attributes of an object, such as a plug-in, to be embedded in a Web page to be viewed with the browser. An example of an object tag is as follows:

```html
<object data="dataLocation" type="MIMEType" align="alignment" height="pixHeight" width="pixWidth" id="name">
<param name="name1" value="value1" />
<param name="name2" value="value2" />
</object>
```

where:

- data is the location of the object's data
  - This is a mandatory attribute.
- type is the MIME type of the plug-in
  - This is a mandatory attribute.
align is the left, right, top, or bottom alignment of the plug-in on the HTML page. This is an optional attribute.

height is the vertical size, in pixels, of the plug-in on the HTML page. This is an optional attribute.

width is the horizontal size, in pixels, of the plug-in on the HTML page. This is an optional attribute.

id is the name of the plug-in. This is an optional attribute.

param name is the name of a parameter required by the plug-in. This is an optional attribute.

value is the initial value of the parameter required by the plug-in. This is an optional attribute.

2.1.2 About the embed tag

The embed tag specifies the attributes of a plug-in to be embedded in a Web page to be viewed with the browser. An example of an embed tag is as follows:

```html
<embed
    src="location"
    type="MIMEtype"
    align="left"|"right"|"top"|"bottom"
    border="borderWidth"
    frameborder="no"
    height="height"
    width="width"
    units="units"
    hspace="horizMargin"
    vspace="vertMargin"
    id="name"
    name1="value1"
    name2="value2"
 />
</embed>
```

where:

- **src** is the URL location of the file to run. This is a mandatory attribute.
- **type** is the MIME type of the plug-in needed to run the file. This is a mandatory attribute.
- **align** is the left, right, top, or bottom alignment of the plug-in on the HTML page. This is an optional attribute.
- **border** is the width, in pixels, of the border surrounding the plug-in on the HTML page. This creates a picture frame effect. This is an optional attribute.
- **frameborder** specifies whether or not the frames on the HTML page appear with borders separating themselves from each other. Values: yes or no. This is an optional attribute.
- **height** is the vertical size of the plug-in on the HTML page. Default unit: pixels.
width is the horizontal size of the plug-in on the HTML page. Default unit: pixels
This is an optional attribute.

units is the unit used for the sizes of the height and width.
For example: inches, cm, mm, point size, or pixels.
This is an optional attribute.

hspace is the width, in pixels, of an invisible border to the left and right of the plug-in on the HTML page.
This creates blank space on the left and right sides of the plug-in object.
This is an optional attribute.

vspace is the width, in pixels, of an invisible border above and below the plug-in on the HTML page.
This creates blank space above and below the plug-in object.
This is an optional attribute.

id is the name of the plug-in
This is an optional attribute.

An embed tag must contain either the src attribute or the type attribute in order for the plug-in to load. The browser uses either the value of the type attribute or the suffix of the file name of the source to determine which plug-in to use. For example:

```html
<embed src="doh.wav" width="100" height="40" type="audio/wav">
</embed>.
```
3 Developing Plug-ins

This chapter provides instructions for developing a plug-in application. A sample plug-in application may be found in source of “mdolphin/plugin_demos/default_plugin” directory.

3.1 Identifying a Plug-in

The browser identifies the following information for each mDolphin plug-in:

- Plug-in name.
- MIME type supported.
- MIME file extensions supported.
- MIME type description.

When the browser needs to display data of a particular MIME type, it finds a plug-in registered to that type and loads the plug-in.

3.2 Register and UnRegister Plug-in

3.2.1 Register a Plug-in to mDolphin

In mDolphin, all Plug_ins should be registered by the following function.

```c
HPGN mdolphin_register_plugin(const PLUGIN_REGISTER * RegPgn);
```

Before calling `mdolphin_register_plugin`, you should fill the `PLUGIN_REGISTER` struct with the plug_in information.

```c
typedef struct _PLUGIN_REGISTER{
    /** Plugin's name. */
    char name[MAX_LEN_PLUGINNAME];
    /** Plugin's mime. */
    char mimetype[MAX_LEN_MIMETYPE];
    /** Plugin's suffixes. */
    char suffixes[MAX_LEN_SUFFIXES];
    /** Plugin's initialize function. (required) */
    NP_InitializeProcPtr init;
    /** Plugin's shutdown function. (required) */
    NP_ShutdownProcPtr shutdown;
    /** Plugin's get mime description function. (can be NULL) */
    NP_GetMIMEDescriptionProcPtr desc;
    /** Plugin's get value function. (can be NULL) */
    NP_GetValueProcPtr getval;
} PLUGIN_REGISTER;
```

3.2.2 UnRegister a Plug-in to mDolphin

mDolphin provides the following function to unregister a plug_in.

```c
void mdolphin_unregister_plugin(HPGN plugin);
```
**Note:** if you want to unregister a specificMimeType plug-in, you can use the following function to get the plug-in’s HPGN. Then call `mdolphin_unregister_plugin`.

```c
HPGN mdolphin_get_plugin_from_mimetype(const char* mimeType);
```

### 3.3 Initializing and Destroying Plug-ins

This section describes the initialization and destruction of Netscape mDolphin plug-in applications.

#### 3.3.1 Plug-in initialization

When a Netscape mDolphin plug-in is initialized, the browser saves the following data:

- Plug-in name.
- MIME type.
- MIME file extension.

The browser passes a table of function pointers to the plug-in. This table is an allocated but uninitialized structure that contains the API that the plug-in provides to the browser.

The plug-in fills out this table during the initialization call. The following code fragment demonstrates the implementation of the `InitializeFuncs` function within a plug-in.

```c
NPError P_NAME(NP_Initialize) (NPNetscapeFuncs* nsTable, NPPluginFuncs* pluginFuncs)
{
    NPError err = NPERR_NO_ERROR;
    /* validate input parameters */
    if ((nsTable == NULL) || (pluginFuncs == NULL))
        err = NPERR_INVALID_FUNCTABLE_ERROR;
    /*
     * Copy all the fields of Netscape function table into our
     * copy so we can call back into Netscape later. Note that
     * we need to copy the fields one by one, rather than assigning
     * the whole structure, because the Netscape function table
     * could actually be bigger than what we expect.
     */
    if (err == NPERR_NO_ERROR) {
        P_NAME(gNetscapeFuncs).size          = nsTable->size;
        P_NAME(gNetscapeFuncs).version       = nsTable->version;
        P_NAME(gNetscapeFuncs).geturlnotify  = nsTable->geturlnotify;
        P_NAME(gNetscapeFuncs).geturl        = nsTable->geturl;
        P_NAME(gNetscapeFuncs).posturlnotify = nsTable->posturlnotify;
        P_NAME(gNetscapeFuncs).posturl       = nsTable->posturl;
        P_NAME(gNetscapeFuncs).requestread   = nsTable->requestread;
        P_NAME(gNetscapeFuncs).newstream     = nsTable->newstream;
        P_NAME(gNetscapeFuncs).write         = nsTable->write;
        P_NAME(gNetscapeFuncs).destroystream = nsTable->destroystream;
        P_NAME(gNetscapeFuncs).status        = nsTable->status;
        P_NAME(gNetscapeFuncs).uagent        = nsTable->uagent;
        P_NAME(gNetscapeFuncs).memalloc      = nsTable->memalloc;
        P_NAME(gNetscapeFuncs).memfree       = nsTable->memfree;
        P_NAME(gNetscapeFuncs).memflush      = nsTable->memflush;
        P_NAME(gNetscapeFuncs).reloadplugins = nsTable->reloadplugins;
        #ifdef OJI
        P_NAME(gNetscapeFuncs).getJavaEnv    = nsTable->getJavaEnv;
        #endif
    }
}
```
P_NAME(gNetscapeFuncs).getJavaPeer = nsTable->getJavaPeer;
#endif
P_NAME(gNetscapeFuncs).getvalue = nsTable->getvalue;
P_NAME(gNetscapeFuncs).setvalue = nsTable->setvalue;
P_NAME(gNetscapeFuncs).invalidateRect = nsTable->invalidateRect;
P_NAME(gNetscapeFuncs).invalidateRegion = nsTable->invalidateRegion;
P_NAME(gNetscapeFuncs).forcedRedraw = nsTable->forcedRedraw;
P_NAME(gNetscapeFuncs).pushpopupEnabledState = nsTable->
pushpopupEnabledState;
P_NAME(gNetscapeFuncs).popuppopupEnabledState = nsTable->
popuppopupEnabledState;
P_NAME(gNetscapeFuncs).enumerate = nsTable->enumerate;
/*
 * Set up the plugin function table that Netscape will use to
 * call us. Netscape needs to know about our version and size
 * and have a UniversalProcPointer for every function we
 * implement.
 */
pluginFuncs->version    = (NP_VERSION_MAJOR << 8) + NP_VERSION_MINOR;
pluginFuncs->size       = sizeof(NPPluginFuncs);
pluginFuncs->newp       = NewNPP_NewProc( P_NAME(Private_New) );
pluginFuncs->destroy    = NewNPP_DestroyProc( P_NAME(Private_Destroy) );
pluginFuncs->setwindow  = NewNPP_SetWindowProc( P_NAME(Private_SetWindow) );
pluginFuncs->newstream  = NewNPP_NewStreamProc( P_NAME(Private_NewStream) );
pluginFuncs->destroystream = NewNPP_DestroyStreamProc( P_NAME(Private_DestroyStream) );
pluginFuncs->createstream = NewNPP_CreateStreamProc( P_NAME(Private_CreateStream) );
pluginFuncs->asfile     = NewNPP_StreamAsFileProc( P_NAME(Private_StreamAsFile) );
pluginFuncs->writeready = NewNPP_WriteReadyProc( P_NAME(Private_WriteReady) );
pluginFuncs->write      = NewNPP_WriteProc( P_NAME(Private_Write) );
pluginFuncs->print      = NewNPP_PrintProc( P_NAME(Private_Print) );
pluginFuncs->event      = NewNPP_HandleEventProc( P_NAME(Private_HandleEvent) );
pluginFuncs->urlnotify  = NewNPP_URLNotifyProc( P_NAME(Private_URLNotify) );
pluginFuncs->getvalue   = NewNPP_GetValueProc( P_NAME(NP_GetValue) );
pluginFuncs->setvalue   = NewNPP_SetValueProc( P_NAME(NP_SetValue) );
#endif OJI
pluginFuncs->javaClass  = NULL;
#endif
P_NAME(pluginLoadCount)++;
//add the counts of the load
if (P_NAME(pluginLoadCount) > 1)
return err;
err = P_NAME(NPP_Initialize) ();
}
return err;

/// This C++ function gets called once when the plugin is loaded,
/// regardless of how many instantiations there is actually playing
/// movies. So this is where all the one time only initialization
/// stuff goes.
NPError
P_NAME(NPP_Initialize) ()
{
    //here you can write your codes, when plug-in initialization
    return NPERR_NO_ERROR;
}

3.3.2 Creating a plug-in instance

The browser calls the NPP_New function to create a plug-in instance. Instance-
pecific private data can be allocated at this time. The following code example shows how to
create a plug-in instance.

// here the plugin creates a plugin instance object which
// will be associated with this newly created NPP instance and
// will do all the necessary job
NPError P_NAME(NPP_New) (NPMIMEType pluginType, NPP instance, uint16 mode, int16 argc,
char* argn[], char* argv[], NPSavedData* saved)
{
    if(instance == NULL)
        return NPERR_INVALID_INSTANCE_ERROR;

    NPError rv = NPERR_NO_ERROR;

    // create a new plugin instance object
    // initialization will be done when the associated window is ready
    mgPluginCreateData ds;
    ds.instance = instance;
    ds.type     = pluginType;
    ds.mode     = mode;
    ds.argc     = argc;
    ds.argn     = argn;
    ds.argv     = argv;
    ds.saved    = saved;

    P_NAME(mgPluginInstanceBase) * plugin = P_NAME(NS_NewPluginInstance) (&ds);
    if(plugin == NULL)
        return NPERR_OUT_OF_MEMORY_ERROR;

    // associate the plugin instance object with NPP instance
    instance->pdata = (void *)plugin;
    return rv;
}

/// Constructor
P_NAME(mgPluginInstance) :: P_NAME(mgPluginInstance) (mgPluginCreateData* data)
:_instance(data->instance)
, m_hWnd(0)
{
    //here you can write your codes, when Creating a plug-in instance
}

3.3.3 Destroying a plug-in instance

The browser calls the NPP_Destroy function to destroy a plug-in instance. The browser application calls the NPP_Destroy function when the user performs any of the following actions:

- Navigates away from the page containing the instance.
- Quits the application.

If this is the last instance created by a plug-in, the browser calls the NPP_Shutdown function. It is important that the plug-in developer deletes all the resources, such as the memory, files, and sockets allocated by the browser (such as streams) before calling the NPP_Destroy function. NPP_Destroy does not track or delete browser-created objects.

The following code example shows how a plug-in instance is deleted.

NPError P_NAME(NPP_Destroy) (NPP instance, NPSavedData** /*save*/)
{
    if(instance == NULL)
        return NPERR_INVALID_INSTANCE_ERROR;

    NPError rv = NPERR_NO_ERROR;

    P_NAME(mgPluginInstanceBase) * plugin = (P_NAME(mgPluginInstanceBase) *)instance->pdata;
if(plugin != NULL)
    P_NAME(NS_DestroyPluginInstance) (plugin);

    return rv;

/// Destructor
P_NAME(mgPluginInstance):: ~P_NAME(mgPluginInstance) ()
{
    //here you can write your codes, when Destroying a plug-in instance
}

3.3.4 Shutdown

The NPP_Shutdown function does the following:

- Informs the plug-in that its library is about to be unloaded.
- Gives the plug-in a chance to perform closing tasks such as:
  - Cancel any outstanding I/O requests
  - Delete threads it created
  - Free any memory it allocated

This function is not called if any existing plug-in instances or plug-in stream instances are open. All plug-in data should be deleted before this call is made. This call is useful when data allocated by the NPP_Initialize function needs to be cleaned up. The following code shows an example of the implementation of the NPP_Shutdown function.

```c
/*
* NP_Shutdown [optional]
* - Netscape needs to know about this symbol.
* - It calls this function after looking up its symbol after
*   the last object of this kind has been destroyed.
* - *
*/
NPError
P_NAME(NP_Shutdown) (void)
{
    NPError err = NPERR_NO_ERROR;
    PLUGINDEBUGSTR("NP_Shutdown");
    P_NAME(pluginLoadCount)--; //sub the counts of the load
    if (P_NAME(pluginLoadCount) == 0)
        P_NAME(NPP_Shutdown) ();
    return err;
}

/// This C++ function gets called once when the plugin is being
/// shutdown, regardless of how many instantiations actually are
/// playing movies. So this is where all the one time only
/// shutdown stuff goes.
void
P_NAME(NPP_Shutdown) ()
{
    //here you can write your codes, when NPP_Shutdown plug-in
3.4 Drawing Plug-ins

In mDolphin, the plug-in application is like MiniGUI application, and supports all the MiniGUI events, so you can draw the plug-in on a Web page, like drawing in MiniGUI.

```c
int16 P_NAME(mgPluginInstance)::HandleEvent( HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
{
    switch(message) {
    case MSG_PAINT:
        RECT rect;
        GetClientRect ( hWnd, &rect);
        HDC hdc;
        hdc = BeginPaint ( hWnd);
        DrawText ( hdc, "not find the useable plugins -- default plugin ",
                 -1, &rect, DT_CENTER);
        EndPaint ( hWnd, hdc);
        return 1;

    case MSG_CREATE:
        m_hWnd = hWnd;
        break;
    }
    return 0; //return 1 : show the plugin handled this message
    //return 0 : show the plugin not handled this message
}
```

3.5 Allocating and Freeing Memory

The plug-in calls the `NPN_MemAlloc` function to dynamically allocate a specified amount of memory. The plug-in calls the `NPN_MemFree` function to de-allocate a block of memory.

```c
/**
 * memAlloc and memFree are implemented by the browser.
 * memFlush has an empty implementation in the browser and does
 * nothing when the plug-in calls this function.
 */
void P_NAME(mgPluginInstance)::TestMemory()
{
    void* pMem = NULL;
    int memLeft = 0;
    // Alloc a zero memory size
    pMem = P_NAME(NPN_MemAlloc) (0);
    // Free the memory
    P_NAME(NPN_MemFree) (pMem);
    // Alloc a small memory size
    pMem = P_NAME(NPN_MemAlloc) (2000);
    // Free the small memory
    P_NAME(NPN_MemFree) (pMem);
    // Alloc a large memory size
    pMem = P_NAME(NPN_MemAlloc) (2000000000);
    // Flush the memory, this function should do nothing
    memLeft = P_NAME(NPN_MemFlush) (2000000000);
    // Free the large memory
    P_NAME(NPN_MemFree) (pMem);
}```
3.6 Implementing Streams

Streams are objects that represent data generated from a URL, or data sent by a plug-in without an associated URL. Streams can be produced by the browser and consumed by a plug-in instance, or produced by a plug-in instance and consumed by the browser. A stream object has an associated MIME type, which identifies the format of the data in the stream. Each stream object is associated with a single plug-in, and a plug-in can hold multiple stream objects.

3.6.1 Sending a stream from the browser to a Plug-in

The browser performs the following tasks when sending a data stream to the plug-in:

1. Creates a stream and informs the plug-in.

To inform a plug-in when a new stream is created, the browser calls the `NPP_NewStream` function. This function also determines which mode the browser should use to send data to the plug-in.

The browser can create a stream for the following types of data:

- File specified in the `src` attribute of the `embed` tag
- Data file
- Full-page instance

2. Finds out from the plug-in how much data it can accept

After calling the `NPP_NewStream` function and before writing data to the plug-in, the browser calls the `NPP_WriteReady` function to determine the maximum number of bytes that the plug-in can accept. This function allows the browser to send only as much data to the plug-in as it can handle at one time, and it helps both the browser and the plug-in to use their resources efficiently.

3. Writes data to the stream object

The browser pushes data into the stream by using a series of calls to the `NPP_WriteReady` and the `NPP_Write` functions. The `NPP_Write` function returns the number of bytes consumed by the plug-in instance. If this is a negative number, the browser calls the `NPP_DestroyStream` function to destroy the stream. If the number returned is smaller than the size of the buffer, the browser sends the remaining data in the
buffer to the plug-in through repeated calls to the \texttt{NPP\_WriteReady} and \texttt{NPP\_Write} functions.

- 4. Notifies the plug-in and deletes the stream

After it sends the stream to the plug-in, the browser calls the \texttt{NPP\_DestroyStream} function whether or not the stream arrived successfully. After the plug-in returns from this function, the browser deletes the NPStream object. The plug-in stores private data associated with the stream in stream->pdata. Any resources that the plug-in allocated for that stream should be deleted when the stream is destroyed. The browser stores private data in stream->ndata. The plug-in should not change the value of ndata.

\textbf{Note: It is not possible to send a data stream from the plug-in to the browser.}

\textbf{3.7 Handling URLs}

A plug-in can request and receive the data associated with any type of URL that the browser can handle.

\textbf{3.7.1 Retrieving data from a URL}

The plug-in calls the \texttt{NPN\_GetURL} function to ask the browser to do one of the following:

- Display data retrieved from a URL in a specified target window or frame
- Deliver the data to the plug-in instance in a new stream

If the browser cannot locate the URL or retrieve the data, it does not create a stream for the plug-in. The developer can call the \texttt{NPN\_GetURLNotify} function to notify the plug-in that the data was not retrieved.

The browser calls the \texttt{NPP\_URLNotify} function to notify the plug-in. The browser then passes the notifyData value to the plug-in. The notifyData parameter contains the private plug-in data passed to the corresponding call to the \texttt{NPN\_GetURLNotify} function. The value of notifyData may be used to track multiple requests.

The \texttt{NPN\_GetURLNotify} function handles the URL request asynchronously. It returns immediately and only later handles the request and calls the \texttt{NPP\_URLNotify} function. The plug-in must receive this notification in order to determine whether a request with a null target failed or whether a request with a non-null target completed successfully.

\textbf{3.7.2 Posting URLs}

The plug-in calls the \texttt{NPN\_PostURL} function to post data from a file or buffer to a URL.
After posting the data, the `NPN_PostURL` function either displays the server response in the target window or delivers it to the plug-in.

The `NPN_PostURLNotify` function has the same capabilities as the `NPN_PostURL` function, with the following exceptions:

- `NPN_PostURLNotify` supports specifying headers when posting a memory buffer
- `NPN_PostURLNotify` calls the `NPP_URLNotify` function upon successful or unsuccessful completion of the request. The `NPN_PostURLNotify` function is asynchronous; it returns immediately and only later handles the request and calls the `NPP_URLNotify` function.

The example of Handling URLs and Implementing Streams see the demo in “mdolphin/plugin_demos/pictureshow-plugin”, it have show how to post a url and receive a stream.
4 Plug-in API Reference Tables

The Browser Plug-in API consists of the following two parts:
- Adaptation of the Netscape Plug-in API for the mDolphin
- Extensions

The browser and plug-ins interact with each other through two interfaces:
- NPN interface—plug-in instances call these to communicate with the browser
- NPP interface—the browser calls these to perform operations on a plug-in

Each function in the API has the prefix NPN or NPP to indicate which interface it uses to communicate.

4.1 Adapted Netscape Plug-in API Functions

The tables in this section contain the functions adapted from the Netscape Plug-in API.

4.1.1 Initialization and destruction functions

The browser calls the functions in this section to initialize or delete a plug-in instance:

4.1.1.1 NP_Initialize

Table 4.1NP_Initialize

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPP — implemented by the plug-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>NPErrror NP_Initialize (NPNetscapeFuncs* NPN, NPPluginFuncs* NPP)</td>
</tr>
<tr>
<td>Parameters</td>
<td>NPNetscapeFuncs *NP: Pointer to the browser’s function table.</td>
</tr>
<tr>
<td></td>
<td>NPPluginFuncs *NPP: Pointer to the plug-in’s function table.</td>
</tr>
</tbody>
</table>
### Returns

NPError status code One of the following:

- 0 means NO_ERROR
- 1 means GENERIC_ERROR
- 2 means INVALID_INSTANCE_ERROR
- 3 means INVALID_FUNCTABLE_ERROR
- 4 means MODULE_LOAD_FAILED_ERROR
- 5 means OUT_OF_MEMORY_ERROR
- 6 means INVALID_PLUGIN_ERROR
- 7 means INVALID_PLUGIN_DIR_ERROR
- 8 means INCOMPATIBLE_VERSION_ERROR
- 9 means INVALID_PARAMETER
- 10 means INVALID_URL
- 11 means FILE_NOT_FOUND
- 12 means NO_DATA
- 13 means STREAM_NOT_SEEKABLE

### Description

Exchanges function tables between the browser and the plug-in.

### 4.1.1.2 NPP_New

Table 4.2 NPP_New

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>Syntax</th>
<th>Parameters</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP — implemented by the plug-in</td>
<td><code>NPError NPP_New(NPMIMEType pluginType, NPP instance, uint16 mode, int16 argc, char* argn[], char* argv[], NPSavedData* saved)</code></td>
<td>NPMIMEType pluginType</td>
<td>The MIME type</td>
</tr>
<tr>
<td>NPP instance</td>
<td></td>
<td>The plug-in instance</td>
<td></td>
</tr>
<tr>
<td>uint16 mode</td>
<td></td>
<td>The mode Value: NP_EMBED</td>
<td></td>
</tr>
<tr>
<td>int16 argc</td>
<td></td>
<td>Numbers of Attribute</td>
<td></td>
</tr>
<tr>
<td>char* argn[]</td>
<td></td>
<td>Attribute of the &lt;object&gt; tag names</td>
<td></td>
</tr>
<tr>
<td>char* argv[]</td>
<td></td>
<td>Attribute of the &lt;object&gt; tag values</td>
<td></td>
</tr>
<tr>
<td>NPSavedData* saved</td>
<td></td>
<td>not supported</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NPError status code</td>
</tr>
</tbody>
</table>

For the status code values, see Table 4.1
Description | Exchanges function tables between the browser and the plug-in.

### 4.1.1.3 NPP_Destroy

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPP — implemented by the plug-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>NPError NPP_Destroy (NPP instance, NPSavedData** save)</code></td>
</tr>
<tr>
<td>Parameters</td>
<td><code>NPP instance</code></td>
</tr>
<tr>
<td></td>
<td><code>NPSavedData* saved</code></td>
</tr>
<tr>
<td>Returns</td>
<td><code>NPError status code</code></td>
</tr>
<tr>
<td></td>
<td>For the status code values, see Table 4.1</td>
</tr>
<tr>
<td>Description</td>
<td>Deletes a plug-in instance.</td>
</tr>
</tbody>
</table>

### 4.1.1.4 NPP_Shutdown

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPP — implemented by the plug-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>void NPP_Shutdown (void)</code></td>
</tr>
<tr>
<td>Parameters</td>
<td>None</td>
</tr>
<tr>
<td>Returns</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Deletes all resources allocated for the plug-in Library.</td>
</tr>
</tbody>
</table>

### 4.1.2 Drawing functions

#### 4.1.2.1 NPN_ForceRedraw

This function forces to redraw all the plug-in view.

#### 4.1.2.2 NPN_InvalidateRect

This function redraws the plug-in’s window rectangle which was given.

#### 4.1.2.3 NPN_InvalidateRegion

We do not implement the Region struct, so you will never need to call it.
### 4.1.2.4 NPN_SetValue
This function has an empty implementation in the browser. If called, this function does nothing.

### 4.1.2.5 NPP_HandleEvent

<table>
<thead>
<tr>
<th>Table 4.5 NPP_HandleEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Plug-in API Type</strong></td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

### 4.1.2.6 NPP_Print
Not supported. The browser never calls this plug-in function.

### 4.1.2.7 NPP_SetValue
Not supported. The browser never calls this plug-in function.

### 4.1.2.8 NPP_SetWindow

<table>
<thead>
<tr>
<th>Table 4.6 NPP_SetWindow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Plug-in API Type</strong></td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
4.1.3 Stream functions

4.1.3.1 NPN_NewStream
This function has an empty implementation in the browser. If called, this function does nothing.

4.1.3.2 NPN_DestroyStream
This function has an empty implementation in the browser, which can be called but does nothing.

4.1.3.3 NPN_RequestRead
This function has an empty implementation in the browser. If called, this function does nothing.

4.1.3.4 NPN_Write
This function has an empty implementation in the browser. If called, this function does nothing.

4.1.3.5 NPP_NewStream

Table 4.7 NPP_NewStream

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPP — implemented by the plug-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>NPP_NewStream(NPP instance, NPMIMEType type, NPStream* stream, NPBool seekable, uint16* stype)</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP instance</td>
<td>The plug-in instance</td>
</tr>
<tr>
<td>NPMIMEType</td>
<td>The MIME type of the stream type</td>
</tr>
<tr>
<td>NPStream*</td>
<td>The new stream object stream</td>
</tr>
<tr>
<td>NPBool</td>
<td>A flag that indicates whether or not the stream is searchable. Searchable streams are not supported. Therefore, the flag is always set to EFalse.</td>
</tr>
<tr>
<td>uint16* stype</td>
<td>The type of the stream. The plug-in should set the stream type. Stream types are:</td>
</tr>
<tr>
<td></td>
<td>- NP_NORMAL,</td>
</tr>
<tr>
<td></td>
<td>- NP_ASFILE</td>
</tr>
<tr>
<td></td>
<td>- NP_ASFILEONLY</td>
</tr>
<tr>
<td></td>
<td>For embed system we just supported NP_NORMAL type.</td>
</tr>
</tbody>
</table>

NP_NORMAL: The plug-in can process the data progressively as it arrives from the network or file system through a series of calls to the NPP_WriteReady and the NPP_Write functions.

### Returns

Returns NPError status code

For the status code values, see Table 2.

### Description

Notifies a plug-in instance of a new data stream.

---

#### 4.1.3.6 NPP_DestroyStream

Table 4.8 NPP_DestroyStream

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP — implemented by the plug-in</td>
<td></td>
</tr>
</tbody>
</table>

Syntax

NPError NPP_DestroyStream (NPP instance, NPStream* stream, NPReason reason)
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP instance</td>
<td>The plug-in instance</td>
</tr>
<tr>
<td>NPStream*</td>
<td>The stream to be destroyed</td>
</tr>
<tr>
<td>NPresReason</td>
<td>The reason for destroying the stream. The reason parameter can have one of the following values:</td>
</tr>
<tr>
<td></td>
<td>NPresDone (Most common) -- normal completion; all data was sent to the instance.</td>
</tr>
<tr>
<td></td>
<td>NPresUserBreak -- the user canceled the stream.</td>
</tr>
<tr>
<td></td>
<td>NPresNetworkErr -- the stream failed because of problems with the network, disk I/O error, lack of memory, or some other problem.</td>
</tr>
</tbody>
</table>

### Returns

Returns NPErrror status code

For the status code values, see Table 2.

### Description

Destroys the stream that was previously created to stream data to the plug-in.

#### 4.1.3.7 NPP_StreamAsFile

Not supported. The browser never calls this plug-in function.

#### 4.1.3.8 NPP_Write

Table 4.9 NPP_Write

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP — implemented by the plug-in</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int32 NPP_Write (NPP instance, NPStream* stream, int32 offset, int32 len, void* buffers)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP instance</td>
<td>The plug-in instance</td>
</tr>
<tr>
<td>NPStream*</td>
<td>The stream</td>
</tr>
<tr>
<td>int32 offset</td>
<td>The offset in the stream.</td>
</tr>
<tr>
<td>int32 len</td>
<td>The size of the new data</td>
</tr>
<tr>
<td>void* buffers</td>
<td>The data itself</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If successful, this function returns the number of bytes consumed by the plug-in instance.</td>
<td></td>
</tr>
<tr>
<td>If unsuccessful, this function destroys the stream by returning a negative value.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writes a chunk of data to the plug-in.</td>
<td></td>
</tr>
</tbody>
</table>
4.1.3.9 NPP_WriteReady

Table 4.10 NPP_WriteReady

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP — implemented by the plug-in</td>
<td></td>
</tr>
</tbody>
</table>

| Syntax | int32 NPP_WriteReady (NPP instance, NPStream* stream) |
| Parameters | NPP instance The plug-in instance |
| NPStream* The stream |
| Returns | The maximum data size that the plug-in can handle. |
| Description | The browser calls the NPP_Write function with the amount of data returned from the NPP_WriteReady function. |

4.1.4 URL functions

4.1.4.1 NPN_GetURL

Table 4.11 NPN_GetURL

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPN — implemented by the browser</td>
<td></td>
</tr>
</tbody>
</table>

| Syntax | NPError NPN_GetURL (NPP instance, const char* url, const char* target) |
| Parameters | NPP instance The plug-in instance |
| const char* url The URL to load |
| const char* target The target window |
| Returns | NPError status code |
| For the status code values, see Table 2. |
| Description | The plug-in calls this function to request the browser to load a URL. |
| Note | If the target window is NULL, pass the response to the plug-in. |
| If the target window is _parent, or _top, the browser initiates a load request to the given URL. |

4.1.4.2 NPN_GetURLNotify

Table 4.12 NPN_GetURLNotify

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPN — implemented by the browser</td>
<td></td>
</tr>
</tbody>
</table>

| Syntax | NPError NPN_GetURLNotify (NPP instance, const char* url, const char* target, void* notifyData) |
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP instance</td>
<td>The plug-in instance</td>
</tr>
<tr>
<td>const char* url</td>
<td>The URL to load</td>
</tr>
<tr>
<td>const char* target</td>
<td>The target window</td>
</tr>
<tr>
<td>void* notifyData</td>
<td>The context to be returned to the plug-in with the notification.</td>
</tr>
</tbody>
</table>

### Returns

NPError status code
For the status code values, see Table 2.

### Description

The plug-in calls this function to request the browser to load a URL. A requesting plug-in is informed when the load completes.

### Note

If the target window is NULL, _parent, or _top, the browser initiates a load request to the given URL. After the load is initiated, the browser notifies the plug-in that the load was successful. There is no way for the browser to ensure that the server received the load request.

---

### 4.1.4.3 NPN_PostURL

Table 4.13 NPN_PostURL

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPN — implemented by the browser</td>
<td></td>
</tr>
</tbody>
</table>

#### Syntax

NPError NPN_PostURL (NPP instance, const char* url, const char* target, const char* buf, NPBool file)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP instance</td>
<td>The plug-in instance</td>
</tr>
<tr>
<td>const char* url</td>
<td>The URL to load</td>
</tr>
<tr>
<td>const char* target</td>
<td>The target window</td>
</tr>
<tr>
<td>const char* buf</td>
<td>A buffer</td>
</tr>
<tr>
<td>NPBool file</td>
<td>A flag indicating the contents of the buffer.</td>
</tr>
<tr>
<td>Value One of the following:</td>
<td></td>
</tr>
<tr>
<td>True indicates that the buffer contains a file name.</td>
<td></td>
</tr>
<tr>
<td>False indicates that the buffer contains the posted data.</td>
<td></td>
</tr>
</tbody>
</table>

**we not support file system operate, so the NPBool file value should always FALSE.**

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPError status code</td>
<td>For the status code values, see Table 2.</td>
</tr>
</tbody>
</table>

| Description | Posts information through the browser and requests that the result be displayed or passed to the named target |
window or frame. If a name is not provided, the target is assumed to be the plug-in itself.

**Note** If the target window is NULL, pass the response to the plug-in. If the target window is _parent, or _top, the browser initiates a load request to the given URL.

### 4.1.4.4 NPN_PostURLNotify

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>NPNError NPN_PostURL (NPP instance, const char* url, const char* target, const char* buf, NPBool file, void* notifyData)</td>
</tr>
<tr>
<td>Parameters</td>
<td>NPP instance The plug-in instance const char* url The URL to load const char* target The target window const char* buf A buffer NPBool file A flag indicating the contents of the buffer. Value One of the following: True indicates that the buffer contains a file name. False indicates that the buffer contains the posted data. We do not support file system operate, so the NPBool file value should always FALSE. void* notifyData The context to be returned to the plug-in with the notification.</td>
</tr>
<tr>
<td>Returns</td>
<td>NPNError status code For the status code values, see Table 2.</td>
</tr>
<tr>
<td>Description</td>
<td>The plug-in calls this function to request the browser to post to a URL. The browser informs the plug-in when the load request is complete.</td>
</tr>
</tbody>
</table>

**Note** If the target window is NULL, pass the response to the plug-in. If the target window is _parent, or _top, the browser initiates a load request to the given URL.
### 4.1.4.5 NPP_URLNotify

Table 4.15: NPN_PostURLNotify

<table>
<thead>
<tr>
<th><strong>Plug-in API Type</strong></th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
<td>NPPError NPP_URLNotify (NPP instance, const char* url, const char* target, void* notifyData)</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>NPP instance The plug-in instance</td>
</tr>
<tr>
<td></td>
<td>const char* url URL of the NPN_GetURLNotify function or of the NPN_PostURLNotify function request</td>
</tr>
<tr>
<td></td>
<td>const char* target Reason code for completion of the request. Values One of the following: NPRES_DONE: Normal completion; all data was sent to the instance. This is the most common value. NPRES_USER BREAK: The user can cancel directly. NPRES_NETWORK_ERR: The stream failed because of problems with the network, disk I/O error, lack of memory, or some other problem.</td>
</tr>
<tr>
<td></td>
<td>void* notifyData Context to be returned to the plug-in with the notification.</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
<td>NPPError status code For the status code values, see Table 2.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Notifies the instance of the completion of a URL request made by the NPN_GetURLNotify function or the NPN_PostURLNotify function.</td>
</tr>
</tbody>
</table>

### 4.1.5 Memory functions

#### 4.1.5.1 NPN_MemAlloc

Table 4.16: NPN_MemAlloc

<table>
<thead>
<tr>
<th><strong>Plug-in API Type</strong></th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
<td>void* NPN_MemAlloc (uint32 size)</td>
</tr>
</tbody>
</table>

---

24
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>uint32 size</th>
<th>The desired memory size</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>The allocated memory.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If this function fails to complete, it returns NULL.</td>
</tr>
</tbody>
</table>

| Description | Allocates memory directly from the operating system on behalf of the plug-in. |

### 4.1.5.2 NPN_MemFlush

This function has an empty implementation in the browser. If called, this function does nothing.

### 4.1.5.3 NPN_MemFree

Table 4.17 NPN_MemFree

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>void NPN_MemFree (void* ptr)</td>
</tr>
<tr>
<td>Parameters</td>
<td>void* ptr A pointer to the memory to be freed.</td>
</tr>
<tr>
<td>Returns</td>
<td>None.</td>
</tr>
<tr>
<td>Description</td>
<td>Frees memory that was previously allocated by the browser.</td>
</tr>
</tbody>
</table>

### 4.1.6 Utility functions

#### 4.1.6.1 NPN_ReloadPlugins

This function has an empty implementation in the browser. If called, this function does nothing.

#### 4.1.6.2 NPN_Status

Table 4.18 NPN_Status

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>void NPN_Status (NPP instance, const char* message)</td>
</tr>
<tr>
<td>Parameters</td>
<td>NPP instance The plug-in instance.</td>
</tr>
<tr>
<td></td>
<td>const char* message The message to display</td>
</tr>
<tr>
<td>Returns</td>
<td>None.</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the current browser status. Displays a small message window.</td>
</tr>
</tbody>
</table>
4.1.6.3 NPN_UserAgent

Table 4.19 NPN_UserAgent

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>const char* NPN_UserAgent(NPP instance)</td>
</tr>
<tr>
<td>Parameters</td>
<td>NPP instance The plug-in instance.</td>
</tr>
<tr>
<td>Returns</td>
<td>The User Agent string configured in the system.</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the currently configured user agent to the plug-in.</td>
</tr>
</tbody>
</table>

4.1.6.4 NPN_Version

This function has an empty implementation in the browser. If called, this function does nothing.

4.1.7 Java communication functions

The Browser Plug-in API does not support Java communication functions.

4.2 Extensions

4.2.1 mdolphin_register_plugin

Table 4.20 mdolphin_register_plugin

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>HPGN mdolphin_register_plugin(const PLUGIN_REGISTER *RegPgn)</td>
</tr>
<tr>
<td>Parameters</td>
<td>Const PLUGIN_REGISTER *RegPgn The plug-in register struct.</td>
</tr>
<tr>
<td>Returns</td>
<td>NULL on fail, non-NULL plugin's handle on success.</td>
</tr>
<tr>
<td>Description</td>
<td>Register a plugin type on mdolphin.</td>
</tr>
</tbody>
</table>

4.2.2 mdolphin_unregister_plugin

Table 4.21 mdolphin_unregister_plugin

| Plug-in API Type | NPN — implemented by the browser |
### 4.2.3 mdolphin_unregister_plugin

#### Table 4.22 mdolphin_unregister_plugin

<table>
<thead>
<tr>
<th>API Type</th>
<th>void mdolphin_unregister_plugin (HPGN(plugin))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>HPGN plugin The plug-in's handle.</td>
</tr>
<tr>
<td>Returns</td>
<td>None.</td>
</tr>
<tr>
<td>Description</td>
<td>Unregister a plugin type.</td>
</tr>
</tbody>
</table>

#### 4.2.3 mdolphin_get_plugin_counts

#### Table 4.22 mdolphin_get_plugin_counts

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>unsigned int mdolphin_get_plugin_counts(void)</td>
</tr>
<tr>
<td>Parameters</td>
<td>None.</td>
</tr>
<tr>
<td>Returns</td>
<td>The number of the plugins.</td>
</tr>
<tr>
<td>Description</td>
<td>The numbers of registered plugins in mdolphin.</td>
</tr>
</tbody>
</table>

#### 4.2.4 mdolphin_get_plugin_from_mimetype

#### Table 4.23 mdolphin_get_plugin_counts

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>HPGN mdolphin_get_plugin_from_mimetype(const char * mimeType)</td>
</tr>
<tr>
<td>Parameters</td>
<td>const char *mimeType The mime type which want to support.</td>
</tr>
<tr>
<td>Returns</td>
<td>NULL on fail ,non-NULL plugin's handle on success.</td>
</tr>
<tr>
<td>Description</td>
<td>Find the registered plugin which support the mimetype.</td>
</tr>
</tbody>
</table>

#### 4.2.5 mdolphin_get_plugin_info

#### Table 4.24 mdolphin_get_plugin_info

<table>
<thead>
<tr>
<th>Plug-in API Type</th>
<th>NPN — implemented by the browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>BOOL mdolphin_get_plugin_info(HPGN plugin, PLUGIN_INFO * plugininfo)</td>
</tr>
<tr>
<td>Parameters</td>
<td>HPGN plugin The plugin's handle.</td>
</tr>
<tr>
<td></td>
<td>PLUGIN_INFO * plugininfo The struct to store the plugin's info.</td>
</tr>
<tr>
<td><strong>Returns</strong></td>
<td>TRUE on success, FALSE on error.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Get the plugin info, and put it in the struct of PluginInfo.</td>
</tr>
</tbody>
</table>

### 4.2.6 mdolphin_get_plugin_info_by_index

Table 4.25 mdolphin_get_plugin_info_by_index

| **Plug-in API Type** | NPN — implemented by the browser |
| **Syntax** | BOOL mdolphin_get_plugin_info_by_index(unsigned int index, PLUGIN_INFO * plugininfo) |
| **Parameters** | unsigned int index. The index of plugin which want to get info (0 base). |
| | PLUGIN_INFO * plugininfo The struct to store the plugin's info. |
| **Returns** | TRUE on success, FALSE on error. |
| **Description** | Get the plugin's info by the index. |

### 4.3 Structures

#### 4.3.1 NPByteRange

This structure is not supported in mDolphin.

#### 4.3.2 NPEmbedPrint

This structure is not supported in mDolphin.

#### 4.3.3 NPFullPrint

This structure is not supported in mDolphin.

#### 4.3.4 NPP

Table 4.26 NPP structure

```c
typedef struct _NPP
{
    void* pdata;
    void* ndata;
} NPP_t;
```
| Parameters | void* pdata | A private value that a plug-in can use to store a pointer to an internal data structure associated with the instance. The browser does not modify this value. |
| Parameters | void* ndata | A private value that the browser uses to store data associated with the plug-in instance. The plug-in should not modify this value. |

| Description | The browser creates an NPP structure for each plug-in instance and passes a pointer to it to the NPP_New function. This pointer identifies the instance on which API calls should operate and represents the opaque instance handle of a plug-in. NPP contains private instance data for both the plug-in and the browser. The NPP_Destroy function informs the plug-in when the NPP instance is about to be deleted. After this call returns, the NPP pointer is no longer valid. |

### 4.3.5 NPPrint
This structure is not supported in mDolphin.

### 4.3.6 NPRect
This structure is define as MiniGUI 's RECT in mDolphin.

### 4.3.7 NPSavedData
This structure is not supported in mDolphin.

### 4.3.8 NPStream

**Table 4.27 NPStream structure**

```c
typedef struct _NPStream
{
    void* pdata;
    void* ndata;
    const char* url;
    uint32 end;
    uint32 lastmodified;
    void* notifyData;
    const char* headers;
} NPStream;
```
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void* pdata</td>
<td>A private value that a plug-in can use to store a pointer to an internal data structure associated with the instance. The browser does not modify this value.</td>
</tr>
<tr>
<td>void* ndata</td>
<td>A private value that the browser uses to store data associated with the plug-in instance. The plug-in should not modify this value.</td>
</tr>
<tr>
<td>const char* url</td>
<td>The URL from which the data in the stream is read or to which the data is written.</td>
</tr>
<tr>
<td>uint32 end</td>
<td>The offset, in bytes, of the end of the stream. This is equivalent to the length of the stream in bytes. This value can be zero for streams of unknown length, such as streams returned from older FTP servers or generated “on the fly” by CGI scripts.</td>
</tr>
<tr>
<td>uint32 lastmodified</td>
<td>The time at which the data in the URL was last modified (if applicable), measured in seconds since 12:00 midnight GMT, January 1, 1970.</td>
</tr>
<tr>
<td>void* notifyData</td>
<td>This parameter is used only for streams generated in response to a NPN_GetURLNotify function or a NPN_PostURLNotify function request. Value: NPN_GetURLNotify function's notifyData parameter value NPN_PostURLNotify function's notifyData parameter value null for other streams</td>
</tr>
<tr>
<td>const char* headers</td>
<td>Response headers from host. Exists only for NPVERS_HAS_RESPONSE_HEADERS. Used for HTTP only; NULL for non-HTTP. Available from NPP_NewStream onwards. Plugin should copy this data before storing it. Includes HTTP status line and all headers, preferably verbatim as received from server, headers formatted as in HTTP (&quot;Header: Value&quot;), and newlines (\n, NOT \r\n) separating them, by \n\0 (NOT \n\n\0).</td>
</tr>
</tbody>
</table>
The browser allocates and initializes the NPStream object and passes it to the plug-in instance as a parameter to the NPP_NewStream function. The browser cannot delete the object until after it calls the NPP_DestroyStream function.

### 4.3.9 NPWindow

**Table 4.28 NPWindow structure**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef struct _NPWindow</td>
<td></td>
</tr>
</tbody>
</table>
| {
<p>| void* window; |
| int32 x; |
| int32 y; |
| uint32 width; |
| uint32 height; |
| NPRect clipRect; |
| void * ws_info; |
| NPWindowType type; |
| } NPWindow; |</p>
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void* window</code></td>
<td>A handle to a native window element.</td>
</tr>
<tr>
<td><code>int32 x</code></td>
<td>The x coordinate of the top left corner of the plug-in relative to the page. The plug-in should not modify this value.</td>
</tr>
<tr>
<td><code>int32 y</code></td>
<td>The y coordinate of the top left corner of the plug-in relative to the page. The plug-in should not modify this value.</td>
</tr>
<tr>
<td><code>uint32 width</code></td>
<td>The width of the plug-in area. The plug-in should not modify this value.</td>
</tr>
<tr>
<td><code>uint32 height</code></td>
<td>The height of the plug-in area. The plug-in should not modify this value.</td>
</tr>
<tr>
<td><code>NPRect clipRect</code></td>
<td>Clipping rectangle in port coordinates, not support in mDolphin.</td>
</tr>
<tr>
<td><code>NPWindowType type</code></td>
<td>Specifies whether the NPWindow instance represents a window or a drawable.</td>
</tr>
<tr>
<td></td>
<td>Values:</td>
</tr>
<tr>
<td></td>
<td><code>NPWindowTypeWindow</code>: Indicates that the window field holds a platform-specific handle to a window. The plug-in is considered windowed.</td>
</tr>
<tr>
<td></td>
<td><code>NPWindowTypeDrawable</code>: Not supported.</td>
</tr>
</tbody>
</table>

In mDolphin, the NPWindowType type is always `NPWindowTypeWindow`. 

**Description**

The NPWindow structure represents the native window. It contains information about coordinate position, size, and some platform-specific information.

A windowed plug-in is drawn into a native window (a native window) on a Web page. For windowed plug-ins, the browser calls the NPP_SetWindow function with an NPWindow structure that represents a drawable (a pointer to an NPWindow allocated by the browser). The window is valid until NPP_SetWindow is called again with a different window or the instance is destroyed.
5 Hello World Plug-in

This chapter will show how to write a simple plug-in on PC. We will write a plug-in which will do nothing, just show "Hello World mDolphin plug-in" on the plug-in window.

5.1 Default Plug-in Demo

In mdolphin/plugin_demos directory, default_plugin demo was provided as a template to describe how to write new plug-ins.

We can run the following command to copy the default_plugin first.

```
cp default_plugin/ hello_plugin -r
```

5.2 Implementing Hello World Plug-in

Enter the hello_plugin/src directory.

5.2.1 Modifing Project Name

Next, modify the Makefile.am file as follows:

```
lib_LTLIBRARIES = libmd_hello_plugin.la
libmd_hello_plugin_la_SOURCES = \
```

Then the plug-in's library name is "libmd_hello_plugin.so".

5.2.2 Define Plug-in Name and MIME Type

We use macro to control the plug-in name and the supported MIME type in mdplugin.h file.

You should modify the mdplugin.h as follows:

```
#define P_NAME(FUNCTION_NAME)   hello_plugin_pr_##FUNCTION_NAME
#define PLUGIN_NAME         "helloworld_plugin"
#define PLUGIN_DESCRIPTION  "helloworld_plugin, the first plugin of mDolphin"
#define MIME_TYPES_HANDLED  "x-minigui/helloworld::"
```

Note:

- About P_NAME

As everyone knows, some embedded systems don’t support the dynamic library, just support static library. And if using static library, functions in plug-in libraries can not have the same name, So we use P_NAME to difference the defined plugin function's name.

- How to compose the MIME_TYPES_HANDLED strings
The string is a list of semicolon separated mimetype specifications. Each mimetype specification consists of three colon separated components. The first component is the mime type itself, the second is a comma separated list of file extensions and the last part is a description string.

For example:

```
"application/x-type1:ext1:A supported mime type;application/x-type2:ext2,ext3,ext4:Another supported mime type"
```

### 5.2.3 Implementing Hello World Plug-in

Plug-in was implemented in `plugin.cpp` and `plugin.h`.

#### 5.2.3.1: Initializing the Plug-in

We can add the plug-in initialization codes at function P::NPP_Initialize() in `plugin.cpp`.

```c
NPError P::NPP_Initialize() {
    printf("the hello world plugin is initialized\n");
    return NPERR_NO_ERROR;
}
```

#### 5.2.3.2: Shutdown the Plug-in

We can add the plugin shutdown codes at function P::NPP_Shutdown() in `plugin.cpp`.

```c
void P::NPP_Shutdown() {
    printf("the hello world plugin is shutdown\n");
}
```

#### 5.2.3.3: Initializing a Plug-in Instance

We can add some codes at function P::mgPluginInstance (mgPluginCreateData* data) in `plugin.cpp` when a new plug-in instance is created.

Here, we just print "A new hello world plugin was created"

```c
P::mgPluginInstance::mgPluginInstance (mgPluginCreateData* data)
:_instance(data->instance)
, _hWnd(0)
{
    printf("A new hello world plugin was created \n");
}
```

#### 5.2.3.4: Destroying a Plug-in Instance

We can add some codes at function ~P::mgPluginInstance () in `plugin.cpp` when
the plug-in instance is destroyed.

Here, we just print "A hello world plug-in was instance destroy"

```c
// Destructor
P_NAME(mgPluginInstance)::~P_NAME(mgPluginInstance)()
{
    printf("A hello world plugin was instance destroy");
}
```

### 5.2.3.5: Handle the Event of Plug-in
After the plug-in instance is created, we can handle the event at function `HandleEvent`.

The plug-in events are like MiniGUI messages, so we can develop a mDolphin plug-in like a MiniGUI's application.

In order to show "Hello World mDolphin plug-in" in the plug-in window, we should handle the `MSG_PAINT` event as follows:

```c
int16 P_NAME(mgPluginInstance)::HandleEvent(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
    case MSG_PAINT:
        RECT rect;
        GetClientRect(hWnd, &rect);
        HDC hdc;
        hdc = BeginPaint(hWnd);
        DrawText(hdc, "Hello World mDolphin plug-in",
                 -1, &rect, DT_CENTER);
        EndPaint(hWnd, hdc);
        return 1;
    case MSG_CREATE:
        m_hWnd = hWnd;
        break;
    }
    return 0; //return 1 : handled message
               //return 0 : not handled message
}
```

**Note:** the `HandleEvent` function return 1 show it had handled the message, or will return 0;

### 5.3 Building and Installing Plug-in

#### 5.3.1 Building Hello World Plug-in
You can run the following command to build hello world plug-in.

```bash
cd hello_plugin
./configure
make
```

#### 5.3.2 Installing Plug-in
Take PC demo for example. After building plug-in successfully, you can find the plug-in library in `src/.libs/` directory.
Copy `libmd_hello_plugin.so` to `app_demos/testpc/.mDolphin/plugins/` directory for finishing plug-in installation.

### 5.4 Write Test Html for Hello World Plug-in

Because "x-minigui/helloworld" MIME type was setted for the hello word plug-in, we should set the embed tag's type is "x-minigui/helloworld".

```html
<html>
<head>
</head>
<body>
<embed src="" width="300" height="200" type="x-minigui/helloworld" />
</body>
</html>
```