|  |  |  |
| --- | --- | --- |
|  |  | **MOTOROLA**  **Technical Note**  Project RhoElements -  NPAPI Plugin Guide  Revision 02 – November 2011 |

**Contents**

[References 6](#_Toc306026936)

[1 Scope 8](#_Toc306026937)

[2 What is an NPAPI plugin 8](#_Toc306026938)

[3 NPAPI types 8](#_Toc306026939)

[3.1 NPClass 8](#_Toc306026940)

[3.1.1 Syntax 9](#_Toc306026941)

[3.1.2 Fields 9](#_Toc306026942)

[3.1.3 Function pointer syntax 10](#_Toc306026943)

[3.2 NPObject 11](#_Toc306026944)

[3.3 NPVariant 11](#_Toc306026945)

[3.4 NPPluginFuncs 13](#_Toc306026946)

[3.5 NPNetscapeFuncs 14](#_Toc306026947)

[3.6 NPP Structure 14](#_Toc306026948)

[4 Implementing an NPAPI plugin 15](#_Toc306026949)

[4.1 Creating plugin entry points 15](#_Toc306026950)

[4.1.1 NP\_GetMimeDescription 16](#_Toc306026951)

[4.1.2 NP\_GetEntryPoints 16](#_Toc306026952)

[4.1.3 NP\_Initialize 17](#_Toc306026953)

[4.1.4 NP\_Shutdown 18](#_Toc306026954)

[4.2 The Example Plugin 19](#_Toc306026955)

[4.3 Implementing MyPlugin class 19](#_Toc306026956)

[4.3.1 Creating a plugin instance 20](#_Toc306026957)

[4.3.2 Destroying a plugin instance 21](#_Toc306026958)

[4.4 Implementing MyObject and MyType class 22](#_Toc306026959)

[4.4.1 Working with attributes and methods of MyObject 22](#_Toc306026960)

[4.4.2 Creating new objects with MyType class 24](#_Toc306026961)

[5 NPAPI reference 24](#_Toc306026962)

[5.1 NPN\_MemAlloc 24](#_Toc306026963)

[5.1.1 Summary 24](#_Toc306026964)

[5.1.2 Syntax 25](#_Toc306026965)

[5.1.3 Parameters 25](#_Toc306026966)

[5.1.4 Returns 25](#_Toc306026967)

[5.1.5 Description 25](#_Toc306026968)

[5.2 NPN\_MemFree 25](#_Toc306026969)

[5.2.1 Summary 25](#_Toc306026970)

[5.2.2 Syntax 25](#_Toc306026971)

[5.2.3 Parameters 26](#_Toc306026972)

[5.2.4 Description 26](#_Toc306026973)

[5.3 NPN\_GetValue 26](#_Toc306026974)

[5.3.1 Summary 26](#_Toc306026975)

[5.3.2 Syntax 26](#_Toc306026976)

[5.3.3 Parameters 26](#_Toc306026977)

[5.3.4 Returns 27](#_Toc306026978)

[5.3.5 Description 27](#_Toc306026979)

[5.4 NPN\_CreateObject 28](#_Toc306026980)

[5.4.1 Summary 28](#_Toc306026981)

[5.4.2 Syntax 28](#_Toc306026982)

[5.4.3 Parameters 28](#_Toc306026983)

[5.4.4 Returns 28](#_Toc306026984)

[5.4.5 Description 28](#_Toc306026985)

[5.5 NPN\_RetainObject 28](#_Toc306026986)

[5.5.1 Summary 28](#_Toc306026987)

[5.5.2 Syntax 28](#_Toc306026988)

[5.5.3 Parameters 29](#_Toc306026989)

[5.5.4 Returns 29](#_Toc306026990)

[5.6 NPN\_ReleaseObject 29](#_Toc306026991)

[5.6.1 Summary 29](#_Toc306026992)

[5.6.2 Syntax 29](#_Toc306026993)

[5.6.3 Parameters 29](#_Toc306026994)

[5.6.4 Returns 29](#_Toc306026995)

[5.7 NPN\_Invoke 29](#_Toc306026996)

[5.7.1 Syntax 29](#_Toc306026997)

[5.7.2 Parameters 30](#_Toc306026998)

[5.7.3 Returns 30](#_Toc306026999)

[5.7.4 Description 30](#_Toc306027000)

[5.8 NPN\_InvokeDefault 30](#_Toc306027001)

[5.8.1 Summary 30](#_Toc306027002)

[5.8.2 Syntax 30](#_Toc306027003)

[5.8.3 Parameters 31](#_Toc306027004)

[5.8.4 Returns 31](#_Toc306027005)

[5.8.5 Description 31](#_Toc306027006)

[5.9 NPN\_Evaluate 31](#_Toc306027007)

[5.9.1 Summary 31](#_Toc306027008)

[5.9.2 Syntax 31](#_Toc306027009)

[5.9.3 Parameters 32](#_Toc306027010)

[5.9.4 Returns 32](#_Toc306027011)

[5.9.5 Description 32](#_Toc306027012)

[5.10 NPN\_GetProperty 32](#_Toc306027013)

[5.10.1 Summary 32](#_Toc306027014)

[5.10.2 Syntax 32](#_Toc306027015)

[5.10.3 Parameters 32](#_Toc306027016)

[5.10.4 Returns 33](#_Toc306027017)

[5.11 NPN\_SetProperty 33](#_Toc306027018)

[5.11.1 Summary 33](#_Toc306027019)

[5.11.2 Syntax 33](#_Toc306027020)

[5.11.3 Parameters 33](#_Toc306027021)

[5.11.4 Returns 33](#_Toc306027022)

[5.12 NPN\_RemoveProperty 34](#_Toc306027023)

[5.12.1 Syntax 34](#_Toc306027024)

[5.12.2 Parameters 34](#_Toc306027025)

[5.12.3 Returns 34](#_Toc306027026)

[5.13 NPN\_HasProperty 34](#_Toc306027027)

[5.13.1 Summary 34](#_Toc306027028)

[5.13.2 Syntax 34](#_Toc306027029)

[5.13.3 Parameters 34](#_Toc306027030)

[5.13.4 Returns 35](#_Toc306027031)

[5.14 NPN\_HasMethod 35](#_Toc306027032)

[5.14.1 Summary 35](#_Toc306027033)

[5.14.2 Syntax 35](#_Toc306027034)

[5.14.3 Parameters 35](#_Toc306027035)

[5.14.4 Returns 35](#_Toc306027036)

[5.15 NPN\_SetException 35](#_Toc306027037)

[5.15.1 Summary 35](#_Toc306027038)

[5.15.2 Syntax 35](#_Toc306027039)

[5.15.3 Parameters 36](#_Toc306027040)

[5.15.4 Returns 36](#_Toc306027041)

[5.16 NPN\_ReleaseVariantValue 36](#_Toc306027042)

[5.16.1 Summary 36](#_Toc306027043)

[5.16.2 Syntax 36](#_Toc306027044)

[5.16.3 Parameters 36](#_Toc306027045)

[5.16.4 Returns 36](#_Toc306027046)

[5.16.5 Description 36](#_Toc306027047)

[5.17 NPN\_GetStringIdentifier 37](#_Toc306027048)

[5.17.1 Summary 37](#_Toc306027049)

[5.17.2 Syntax 37](#_Toc306027050)

[5.17.3 Parameters 37](#_Toc306027051)

[5.17.4 Returns 37](#_Toc306027052)

[5.17.5 Description 37](#_Toc306027053)

[5.18 NPN\_GetStringIdentifiers 37](#_Toc306027054)

[5.18.1 Summary 37](#_Toc306027055)

[5.18.2 Syntax 37](#_Toc306027056)

[5.18.3 Parameters 37](#_Toc306027057)

[5.18.4 Returns 38](#_Toc306027058)

[5.18.5 Description 38](#_Toc306027059)

[5.19 NPN\_GetIntIdentifier 38](#_Toc306027060)

[5.19.1 Summary 38](#_Toc306027061)

[5.19.2 Syntax 38](#_Toc306027062)

[5.19.3 Parameters 38](#_Toc306027063)

[5.19.4 Returns 38](#_Toc306027064)

[5.19.5 Description 38](#_Toc306027065)

[5.20 NPN\_IdentifierIsString 39](#_Toc306027066)

[5.20.1 Summary 39](#_Toc306027067)

[5.20.2 Syntax 39](#_Toc306027068)

[5.20.3 Parameters 39](#_Toc306027069)

[5.20.4 Returns 39](#_Toc306027070)

[5.21 NPN\_UTF8FromIdentifier 39](#_Toc306027071)

[5.21.1 Summary 39](#_Toc306027072)

[5.21.2 Syntax 39](#_Toc306027073)

[5.21.3 Parameters 39](#_Toc306027074)

[5.21.4 Returns 39](#_Toc306027075)

[5.21.5 Description 40](#_Toc306027076)

[5.22 NPN\_IntFromIdentifier 40](#_Toc306027077)

[5.22.1 Summary 40](#_Toc306027078)

[5.22.2 Syntax 40](#_Toc306027079)

[5.22.3 Parameters 40](#_Toc306027080)

[5.22.4 Returns 40](#_Toc306027081)

[5.23 NPP\_New 40](#_Toc306027082)

[5.23.1 Summary 40](#_Toc306027083)

[5.23.2 Syntax 40](#_Toc306027084)

[5.23.3 Parameters 41](#_Toc306027085)

[5.23.4 Returns 41](#_Toc306027086)

[5.23.5 Description 41](#_Toc306027087)

[5.24 NPP\_Destroy 42](#_Toc306027088)

[5.24.1 Summary 42](#_Toc306027089)

[5.24.2 Syntax 42](#_Toc306027090)

[5.24.3 Parameters 42](#_Toc306027091)

[5.24.4 Returns 42](#_Toc306027092)

[5.24.5 Description 42](#_Toc306027093)

[5.25 NPP\_SetWindow 43](#_Toc306027094)

[5.25.1 Summary 43](#_Toc306027095)

[5.25.2 Syntax 43](#_Toc306027096)

[5.25.3 Parameters 43](#_Toc306027097)

[5.25.4 Returns 43](#_Toc306027098)

[5.25.5 Description 43](#_Toc306027099)

[5.26 NPP\_NewStream 44](#_Toc306027100)

[5.26.1 Summary 44](#_Toc306027101)

[5.26.2 Syntax 44](#_Toc306027102)

[5.26.3 Parameters 44](#_Toc306027103)

[5.26.4 Returns 45](#_Toc306027104)

[5.26.5 Description 45](#_Toc306027105)

[5.27 NPN\_DestroyStream 46](#_Toc306027106)

[5.27.1 Summary 46](#_Toc306027107)

[5.27.2 Syntax 46](#_Toc306027108)

[5.27.3 Parameters 46](#_Toc306027109)

[5.27.4 Returns 47](#_Toc306027110)

[5.27.5 Description 47](#_Toc306027111)

[5.28 NPN\_StreamAsFile 47](#_Toc306027112)

[5.28.1 Summary 47](#_Toc306027113)

[5.28.2 Syntax 47](#_Toc306027114)

[5.28.3 Parameters 47](#_Toc306027115)

[5.28.4 Description 48](#_Toc306027116)

[5.29 NPP\_WriteReady 48](#_Toc306027117)

[5.29.1 Summary 48](#_Toc306027118)

[5.29.2 Syntax 48](#_Toc306027119)

[5.29.3 Parameters 48](#_Toc306027120)

[5.29.4 Returns 48](#_Toc306027121)

[5.29.5 Description 48](#_Toc306027122)

[5.30 NPP\_Write 49](#_Toc306027123)

[5.30.1 Summary 49](#_Toc306027124)

[5.30.2 Syntax 49](#_Toc306027125)

[5.30.3 Parameters 49](#_Toc306027126)

[5.30.4 Returns 49](#_Toc306027127)

[5.30.5 Description 50](#_Toc306027128)

[5.31 NPP\_Print 50](#_Toc306027129)

[5.31.1 Summary 50](#_Toc306027130)

[5.31.2 Syntax 50](#_Toc306027131)

[5.31.3 Parameters 50](#_Toc306027132)

[5.31.4 Description 50](#_Toc306027133)

[5.32 NPP\_HandleEvent 51](#_Toc306027134)

[5.32.1 Summary 51](#_Toc306027135)

[5.32.2 Syntax 51](#_Toc306027136)

[5.32.3 Parameters 51](#_Toc306027137)

[5.32.4 Returns 52](#_Toc306027138)

[5.32.5 Description 52](#_Toc306027139)

[5.33 NPP\_URLNotify 52](#_Toc306027140)

[5.33.1 Summary 52](#_Toc306027141)

[5.33.2 Syntax 52](#_Toc306027142)

[5.33.3 Parameters 52](#_Toc306027143)

[5.33.4 Description 53](#_Toc306027144)

[5.34 NPP\_GetValue 53](#_Toc306027145)

[5.34.1 Summary 53](#_Toc306027146)

[5.34.2 Syntax 53](#_Toc306027147)

[5.34.3 Parameters 53](#_Toc306027148)

[5.34.4 Returns 54](#_Toc306027149)

[5.34.5 Description 54](#_Toc306027150)

[5.35 NPP\_SetValue 54](#_Toc306027151)

[5.35.1 Summary 54](#_Toc306027152)

[5.35.2 Syntax 54](#_Toc306027153)

[5.35.3 Parameters 54](#_Toc306027154)

[5.35.4 Returns 54](#_Toc306027155)

[5.35.5 Description 55](#_Toc306027156)

[5.36 Error Codes 55](#_Toc306027157)

# References

1. <http://colonelpanic.net/>
2. <https://developer.mozilla.org>
3. <http://en.wikipedia.org>

Document History

| Ref. | Contributors | Date | Description |
| --- | --- | --- | --- |
| Rev 1 | Dinka Ranns –cdhw84 | October 2011 | Initial document creation |
| Rev 2 | Dinka Ranns –cdhw84 | November 2011 | In chapter 4.1.1 added a note about how the RhoElement webkit looks for MIMEType string in the resource file. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Scope

This document describes one of the possible ways of implementing an NPAPI plugin which will expose custom Javascript object on the webpage. It is supposed to offer only a guideline for developing a custom NPAPI plugin for RhoElements and reader is strongly encouraged to check the signature of every function mentioned on the Mozzila Development documentation (MDN docs) that was ,at the time this document was written, located here <https://developer.mozilla.org>

To aid with understanding of NPAPI interface , this document borrows heavily from MDN docs when it comes to function/ object description.

# What is an NPAPI plugin

An NPAPI plugin is a library that interacts with the browser using the Netscape Plugin Application Programming Interface (NPAPI). The original interface was designed for the Netscape family of browsers, but was later adopted by many other browsers in use today. [3]

The NPAPI interface consists of functions exposed by the browser and those exposed by the plugin. Names of functions exposed by the browser begin with NPN\_ (N for Netscape) and those exposed by the plugin with NPP\_ (P for plugin). For the full list of NPAPI functions, please check MDN docs.

One thing to keep in mind is that the plugin operates in the browser’s memory space. It is very important to keep a close eye on the memory management in the plugin, as sometimes it is necessary to request memory from the browser or to inform the browser that a certain reference is no longer in use by the plugin. These sorts of memory interactions are not always obvious and the developer is strongly encouraged to check the NPAPI documentation for details on memory management. For example, when NPN\_GetValue functions is used to retrieve an NPObject, then the plugin needs to call NPN\_ReleaseObjects once it is done using the retrieved NPObject.[1]

# NPAPI types

## NPClass

NPClass is a structure that holds a set of function pointers which make up the

behaviour of an instance of an NPClass [2]. These are the functions the browser (and sometimes plugin) will use to interact with the objects exposed by the plugin. Once an NPClass is exposed to the browser using an NPObject, NPN\_x functions will be mapped to the function pointers in the class. For example, invoking NPN\_HasMethod will in turn call the function mapped to the NPHasMethodFunctionPtr member.

The function pointers in the NP\_Class should not be called directly. Instead, appropriate NPN\_ functions should be invoked.[2]

Below is a description of NPClass from MDN docs at the time this document was written.

### Syntax

struct NPClass

{

uint32\_t structVersion;

NPAllocateFunctionPtr allocate;

NPDeallocateFunctionPtr deallocate;

NPInvalidateFunctionPtr invalidate;

NPHasMethodFunctionPtr hasMethod;

NPInvokeFunctionPtr invoke;

NPInvokeDefaultFunctionPtr invokeDefault;

NPHasPropertyFunctionPtr hasProperty;

NPGetPropertyFunctionPtr getProperty;

NPSetPropertyFunctionPtr setProperty;

NPRemovePropertyFunctionPtr removeProperty;

NPEnumerationFunctionPtr enumerate;

NPConstructFunctionPtr construct;

};

### Fields

**structVersion**

The version number of the structure. This is set to NP\_CLASS\_STRUCT\_VERSION, which is 1 in Mozilla 1.8.\*, 2 since Mozilla 1.9a1, and 3 since Firefox 3.0b1.

**allocate**

Returns a pointer to a newly allocated NPObject. Called by NPN\_CreateObject() if non-NULL, otherwise the browser calls malloc(). This function is expected to allocate and return enough storage to hold the NPObject that is being created.

**deallocate**

Called by NPN\_ReleaseObject() when an object's reference count reaches zero. If this field is NULL, free() is called instead.

**invalidate**

Called on live objects that belong to a plugin instance that is being destroyed. This call is always followed by a call to the deallocate function, or free(). Any attempt to use an invalidated object will result in undefined behavior.

**hasMethod**

Called by NPN\_HasMethod() to determine whether or not a specified method exists on a given NPObject. Returns true if the method exists, otherwise returns false.

**invoke**

Called by NPN\_Invoke() to invoke a specific method on a given NPObject. Returns true if invocation succeeded or false if an error occurred.

**invokeDefault**

Called by NPN\_InvokeDefault() to invoke the default method (if available) on a given NPObject. Returns true if invocation succeeded or false if an error occurred.

**hasProperty**

Called by NPN\_HasProperty() to check whether a given property exists on a given NPObject. Returns true if the specified property exists, otherwise returns false.

**getProperty**

Called by NPN\_GetProperty() to get the value of the specified property on a given NPObject. Returns true if the value was successfully retrieved, otherwise returns false.

**setProperty**

Called by NPN\_SetProperty() to set the value of the specified property on a given NPObject. Returns true if the value was successfully set, otherwise returns false.

**removeProperty**

Called by NPN\_RemoveProperty() to remove a given property from a specified NPObject. Returns true if the property was successfully removed, otherwise returns false.

**enumerate**

Called by NPN\_Enumerate() . This field is available only if structVersion is NP\_CLASS\_STRUCT\_VERSION\_ENUM (2) or greater.

**construct**

Called by NPN\_Construct() . This field is available only if structVersion is NP\_CLASS\_STRUCT\_VERSION\_CTOR (3) or greater.

### Function pointer syntax

typedef NPObject \*(\*NPAllocateFunctionPtr)(NPP npp, NPClass \*aClass);

typedef void (\*NPDeallocateFunctionPtr)(NPObject \*npobj);

typedef void (\*NPInvalidateFunctionPtr)(NPObject \*npobj);

typedef bool (\*NPHasMethodFunctionPtr)(NPObject \*npobj,

NPIdentifier name);

typedef bool (\*NPInvokeFunctionPtr)(NPObject \*npobj,

NPIdentifier name,

const NPVariant \*args,

uint32\_t argCount,

NPVariant \*result);

typedef bool (\*NPInvokeDefaultFunctionPtr)(NPObject \*npobj,

const NPVariant \*args,

uint32\_t argCount,

NPVariant \*result);

typedef bool (\*NPHasPropertyFunctionPtr)(NPObject \*npobj,

NPIdentifier name);

typedef bool (\*NPGetPropertyFunctionPtr)(NPObject \*npobj,

NPIdentifier name,

NPVariant \*result);

typedef bool (\*NPSetPropertyFunctionPtr)(NPObject \*npobj,

NPIdentifier name,

const NPVariant \*value);

typedef bool (\*NPRemovePropertyFunctionPtr)(NPObject \*npobj,

NPIdentifier name);

typedef bool (\*NPEnumerationFunctionPtr)(NPObject \*npobj,

NPIdentifier \*\*value,

uint32\_t \*count);

typedef bool (\*NPConstructFunctionPtr)(NPObject \*npobj,

const NPVariant \*args,

uint32\_t argCount,

NPVariant \*result);

## NPObject

One of the main types in NPAPI is an NPObject. It is used for objects exposed by the plugin or by the browser. For example, the browser will use an NPObject to expose a window object to the plugin.

In RhoElements, NPObjects are used to provide access to hardware functionality in JavaScript. For example, ‘Timer’ is an NPObject embedded on the page which can be used to access Timer functionality in RhoElements.

NPobject structure looks like this :

struct NPObject {

NPClass \*\_class;

uint32\_t referenceCount;

};

Where \_class is a pointer to the NPClass to which the NPobject belongs to. It is recommended not to manipulate the \_class and referenceCount fields directly. Instead, appropriate NPN functions should be used. [2]

Functions that are related to NPObject are :

* [NPN\_CreateObject](#_NPN_CreateObject)()
* [NPN\_RetainObject](#_NPN_RetainObject)()
* [NPN\_ReleaseObject](#_NPN_ReleaseObject)()
* [NPN\_Invoke](#_NPN_Invoke)()
* [NPN\_InvokeDefault](#_NPN_InvokeDefault)()
* [NPN\_Evaluate](#_NPN_Evaluate)()
* [NPN\_GetProperty](#_NPN_GetProperty)()
* [NPN\_SetProperty](#_NPN_SetProperty)()
* [NPN\_RemoveProperty](#_NPN_RemoveProperty)()
* [NPN\_HasProperty](#_NPN_HasProperty)()
* [NPN\_HasMethod](#_NPN_HasMethod)()
* [NPN\_SetException](#_NPN_SetException)()

## NPVariant

NPVariant is a struct that holds a value and a type of that value. The value is held in a union, and the type is one of the types defined in the NPVariantType enumeration.

In RhoElements NPAPI, NPVariants are used to pass arguments to methods, for return values from method invocations or for setting a property value. For example, in a call to NPN\_SetProperty, the value to which the property needs to be set is passed in as an NPVariant.

NPVariant structure looks like this :

typedef struct \_NPVariant {

NPVariantType type;

union {

bool boolValue;

int32\_t intValue;

double\_t doubleValue;

NPString stringValue;

NPObject \*objectValue;

} value;

} NPVariant;

The type field specifies the data contained in the NPVariant, and the value field contains the value stored in the NPVariant.

When using **NPVariant**s to access JavaScript objects in the browser, or vice versa, the mapping of JavaScript values to NPVariants is as follows:

|  |  |
| --- | --- |
| JavaScript type | NPVariantType |
| undefined | NPVariantType\_Void |
| null | NPVariantType\_Null |
| boolean | NPVariantType\_Bool |
| number | NPVariantType\_Int32 or NPVariantType\_Double |
| string | NPVariantType\_String |
| All other types | NPVariantType\_Object |

Functions that manipulate NPVariant are :

* [NPN\_ReleaseVariantValue()](#_NPN_ReleaseVariantValue)
* [NPN\_GetStringIdentifier()](#_NPN_GetStringIdentifier)
* [NPN\_GetStringIdentifiers()](#_NPN_GetStringIdentifiers)
* [NPN\_GetIntIdentifier()](#_NPN_GetIntIdentifier)
* [NPN\_IdentifierIsString()](#_NPN_IdentifierIsString)
* [NPN\_UTF8FromIdentifier()](#_NPN_UTF8FromIdentifier)
* [NPN\_IntFromIdentifier()](#_NPN_IntFromIdentifier)

Plugin developers are not expected to directly manipulate or access the members of the NPVariant instance. To do this, you should use functions above and the following macros :

NPVARIANT\_IS\_VOID()

Evaluates to true if v is of type NPVariantType\_Void.

NPVARIANT\_IS\_NULL()

Evaluates to true if v is of type NPVariantType\_Null.

NPVARIANT\_IS\_BOOLEAN()

Evaluates to true if v is of type NPVariantType\_Bool.

NPVARIANT\_IS\_INT32()

Evaluates to true if v is of type NPVariantType\_Int32.

NPVARIANT\_IS\_DOUBLE()

Evaluates to true if v is of type NPVariantType\_Double.

NPVARIANT\_IS\_STRING()

Evaluates to true if v is of type NPVariantType\_String.

NPVARIANT\_IS\_OBJECT()

Evaluates to true if v is of type NPVariantType\_Object.

NPVARIANT\_TO\_BOOLEAN()

Extracts the boolean value from v.

NPVARIANT\_TO\_INT32()

Extracts a signed 32-bit integer value from v.

NPVARIANT\_TO\_DOUBLE()

Extracts a double precision floating point value from v.

NPVARIANT\_TO\_STRING()

Extracts the NPString value from v.

NPVARIANT\_TO\_OBJECT()

Extracts the NPObject value from v.

VOID\_TO\_NPVARIANT()

Initialize v to a variant of type NPVariantType\_Void.

NULL\_TO\_NPVARIANT()

Initialize v to a variant of type NPVariantType\_Null.

BOOLEAN\_TO\_NPVARIANT()

Initialize v to a variant of type NPVariantType\_Bool with the value val.

INT32\_TO\_NPVARIANT()

Initialize v to a variant of type NPVariantType\_Int32 with the value val.

DOUBLE\_TO\_NPVARIANT()

Initialize v to a variant of type NPVariantType\_Double with the value val.

STRINGZ\_TO\_NPVARIANT()

Initialize v to a variant of type NPVariantType\_String with the value being an NPString holding the UTF-8 string value val.

STRINGN\_TO\_NPVARIANT()

Initialize v to a variant of type NPVariantType\_String with the value being an NPString holding the UTF-8 string value val with the length len.

OBJECT\_TO\_NPVARIANT()

Initialize v to a variant of type NPVariantType\_Object with the value val.

## NPPluginFuncs

NPPlugin func is a structure that contains function pointers the browser will use to interact with the plugin. It is a counterpart of the NPNetscapeFuncs structure we will talk about later in chapter 3.5.

The NPPluginFuncs contains pointers to the following functions

[NPP\_New](#_NPP_New)

[NPP\_Destroy](#_NPP_Destroy)

[NPP\_SetWindow](#_NPP_SetWindow)

[NPP\_NewStream](#_NPP_NewStream)

[NPP\_DestroyStream](#_NPN_DestroyStream)

[NPP\_StreamAsFile](#_NPN_StreamAsFile)

[NPP\_WriteReady](#_NPP_WriteReady)

[NPP\_Write](#_NPP_Write)

[NPP\_Print](#_NPP_Print)

[NPP\_HandleEvent](#_NPP_HandleEvent)

[NPP\_URLNotify](#_NPP_URLNotify)

[NPP\_GetValue](#_NPP_GetValue)

[NPP\_SetValue](#_NPP_SetValue)

Most of the plugin’s functionality will be contained in the implementation of these functions.

## NPNetscapeFuncs

NPNetscape Funcs structure contains function pointers to the functions exposed by the browser. The plugin will use these pointers to allocate/deallocate memory, get browser objects, expose objects on the page etc.

The browser will pass in its NPNetscapeFuncs structure to the plugin at initialization point and it is up to the plugin to store it for later use.

In our example plugin, we will be using the following functions:

[NPN\_MemAlloc](#_NPN_MemAlloc)

[NPN\_MemFree](#_NPN_MemFree)

[NPN\_GetValue](#_NPN_GetValue)

[NPN\_GetStringIdentifier](#_NPN_GetStringIdentifier)

[NPN\_UTF8FromIdentifier](#_NPN_UTF8FromIdentifier)

[NPN\_CreateObject](#_NPN_CreateObject)

[NPN\_RetainObject](#_NPN_RetainObject)

[NPN\_ReleaseObject](#_NPN_ReleaseObject)

[NPN\_SetProperty](#_NPN_SetProperty)

For information on other browser exposed functions, consult MDN Docs.

## NPP Structure

NPP structure represents a plugin instance. It is used as parameter to many NPAPI functions to identify the instance of the plugin that needs to handle the call. The structure contains both plugin’s and browser’s private data. It will be first passed in to the plugin in a call to NPP\_New where a plugin is given the choice to set its own private data. There is no definition of what this private data should look like, you can set it to whatever data will be useful to keep for a plugin Instance. For example it can be a pointer to the plugin instance object.

The structure looks like this :

typedef struct \_NPP

{

void\* pdata; /\* plugin’s private data \*/

void\* ndata; /\* browser’s private data \*/

} NPP\_t;

typedef NPP\_t\* NPP;

# Implementing an NPAPI plugin

## Creating plugin entry points

The browser starts the communication with the plugin using the following functions in the NPAPI plugin

NP\_GetMimeDescription

NP\_GetEntryPoints

NP\_Initialize

NP\_Shutdown

These functions need to be exported from the plugin. In windows Visual Studio it can be done by adding the following in the project .def file :

EXPORTS

NP\_GetEntryPoints @1

NP\_Initialize @2

NP\_Shutdown @3

On Android, add something like this to one of your header files :

#define EXPORT \_\_attribute\_\_((visibility("default")))

extern "C"

{

EXPORT NPError NP\_Initialize(NPNetscapeFuncs\* browserFuncs, NPPluginFuncs\* pluginFuncs, void \*java\_env);

EXPORT const char\* NP\_GetMIMEDescription(void);

EXPORT NPError NP\_Shutdown(void);

/\*NP\_GetValue will not be used in our example. However we will touch on it a

\* bit later

\*/

EXPORT NPError NP\_GetValue(NPP instance, NPPVariable variable, void \*value);

};

You will notice that different functions are exported in the Windows .def file used for our Windows plugin example and in the header file used for our Android example. This is because the initialization process of a plugin varies between Android browser and Windows browser the author used. Before assuming how the browser used will behave, the reader is strongly advised to experiment what functions are invoked and in which order.

Next step is to implement these functions.

### NP\_GetMimeDescription

On Android, NP\_GetMimeDescription is used to get the list of Mime types supported by this plugin.

The return value from the function is a string containing the list of MIME type descriptions. Each MIME type description contains the MIME Type, an extensions list and a short description separated by a colon “:” In the example below, the MIME type is ‘application/x-wtg-jsobjects’, there are no extensions applicable to this plugin and the description is ‘this plugin is grrreat’

If you want to support more than one MIME type, each MIME type description should be separated by a semicolon (;).

const char \*NP\_GetMIMEDescription(void)

{

return "application/x-wtg-jsobjects:: this plugin is grrreat";

}

On Windows, you will not see this function invoked. Instead, set the MIMEType property in your .rc project file to the name of the MIME type your plugin support. If the plugin supports more than one MIME type , separate the values with ‘|’. For example, this is an extract from an .rc file for a plugin that supports three MIME types :

BEGIN

BLOCK "StringFileInfo"

BEGIN

BLOCK "040904e4"

BEGIN

VALUE "MIMEType", "application/x-first-mime |application/x-second-mime |application/x-third-mime"

…

END

END

…

END

Note : At the moment, RhoElements webkit is only looking in the sub-block "\StringFileInfo\040904E4” of the plugin resource file to find the MIMEType resource. Character set combination “040904E4” corresponds to language set to “US English” and the character set to “Windows Multilingual" .

### NP\_GetEntryPoints

NP\_GetEntryPoints function seems to only be used on Windows. This function is invoked by the browser to retrieve the plugin’s [NPPluginFuncs](#_NPPluginFuncs) structure.

NPError OSCALL NP\_GetEntryPoints(NPPluginFuncs\* pFuncs)

{

if(pFuncs == NULL)

return NPERR\_INVALID\_FUNCTABLE\_ERROR;

if(pFuncs->size < sizeof(NPPluginFuncs))

return NPERR\_INVALID\_FUNCTABLE\_ERROR;

MyPlugin::getEntryPoints(pFuncs); //setting browsers NPPluginFuncs to point to

//functions in MyPlugin class

return NPERR\_NO\_ERROR;

}

…

void MyPlugin::getEntryPoints(NPPluginFuncs\* pFuncs)

{

pFuncs->newp = &MyPlugin::NPP\_New;

pFuncs->destroy = &MyPlugin::NPP\_Destroy;

pFuncs->setwindow = &MyPlugin::NPP\_SetWindow;

pFuncs->newstream = &MyPlugin::NPP\_NewStream;

pFuncs->destroystream = &MyPlugin::NPP\_DestroyStream;

pFuncs->asfile = &MyPlugin::NPP\_StreamAsFile;

pFuncs->writeready = &MyPlugin::NPP\_WriteReady;

pFuncs->write = &MyPlugin::NPP\_Write;

pFuncs->print = &MyPlugin::NPP\_Print;

pFuncs->event = &MyPlugin::NPP\_HandleEvent;

pFuncs->urlnotify = &MyPlugin::NPP\_URLNotify;

pFuncs->getvalue = &MyPlugin::NPP\_GetValue;

pFuncs->setvalue = &MyPlugin::NPP\_SetValue;

pFuncs->javaClass = NULL;

}

On Android, this structure is retrieved in the call to NP\_Initialize.

### NP\_Initialize

MDN docs say that this is the first function that the browser calls. However, it has been observed that, on Windows, the NP\_GetEntryPoints will be invoked before NP\_Initialize.[1]

The browser will call NP\_Initialize function when the plug-in is loaded and before the first instance is created; any global plugin initialization should be done at this point.

The signature of this function differs between Android and Windows.

On Android, this function looks like this :

NPError NP\_Initialize(NPNetscapeFuncs\* nFuncs, NPPluginFuncs\* pFuncs, void \*java\_env)

Where nFuncs is a pointer to the browser’s NPNetscapeFuncs struct, pFunc is a pointer to the plugin’s NPPluginFuncs structure (see [NP\_GetEntryPoints](#_NP_GetEntryPoints) ) and java\_env is a pointer to the Java environment for the plugin apk. The use of this pointer is beyond the scope of this document. For more information, check NDK related documentation.

On Windows, the signature of the function is:

NPError OSCALL NP\_Initialize(NPNetscapeFuncs\* nFuncs)

As on Android, nFuncs is a pointer to the browser’s NPNetscapeFuncs struct. We need to store this structure in our plugin for later use.

A typical Windows implementation of this function would look something like this :

NPError OSCALL NP\_Initialize(NPNetscapeFuncs\* nFuncs)

{

if(nFuncs == NULL)

return NPERR\_INVALID\_FUNCTABLE\_ERROR;

if(HIBYTE(nFuncs->version) > NP\_VERSION\_MAJOR)

return NPERR\_INCOMPATIBLE\_VERSION\_ERROR;

if(nFuncs->size < offsetof(NPNetscapeFuncs, setexception) +

sizeof(NPN\_SetExceptionProcPtr))

return NPERR\_INVALID\_FUNCTABLE\_ERROR;

MyPlugin::setBrowserFuncs(nFuncs); //initializing plugins NPNetscapeFuncs

//structure

return NPERR\_NO\_ERROR;

}

…

void MyPlugin::setBrowserFuncs(NPNetscapeFuncs\* nFuncs)

{

NPNFuncs.size = nFuncs->size;

NPNFuncs.version = nFuncs->version;

NPNFuncs.geturlnotify = nFuncs->geturlnotify;

NPNFuncs.geturl = nFuncs->geturl;

NPNFuncs.posturlnotify = nFuncs->posturlnotify;

NPNFuncs.posturl = nFuncs->posturl;

NPNFuncs.requestread = nFuncs->requestread;

NPNFuncs.newstream = nFuncs->newstream;

NPNFuncs.write = nFuncs->write;

NPNFuncs.destroystream = nFuncs->destroystream;

NPNFuncs.status = nFuncs->status;

NPNFuncs.uagent = nFuncs->uagent;

NPNFuncs.memalloc = nFuncs->memalloc;

NPNFuncs.memfree = nFuncs->memfree;

NPNFuncs.memflush = nFuncs->memflush;

NPNFuncs.reloadplugins = nFuncs->reloadplugins;

NPNFuncs.getJavaEnv = nFuncs->getJavaEnv;

NPNFuncs.getJavaPeer = nFuncs->getJavaPeer;

NPNFuncs.getvalue = nFuncs->getvalue;

NPNFuncs.setvalue = nFuncs->setvalue;

NPNFuncs.invalidaterect = nFuncs->invalidaterect;

NPNFuncs.invalidateregion = nFuncs->invalidateregion;

NPNFuncs.forceredraw = nFuncs->forceredraw;

NPNFuncs.getstringidentifier = nFuncs->getstringidentifier;

NPNFuncs.getstringidentifiers = nFuncs->getstringidentifiers;

NPNFuncs.getintidentifier = nFuncs->getintidentifier;

NPNFuncs.identifierisstring = nFuncs->identifierisstring;

NPNFuncs.utf8fromidentifier = nFuncs->utf8fromidentifier;

NPNFuncs.intfromidentifier = nFuncs->intfromidentifier;

NPNFuncs.createobject = nFuncs->createobject;

NPNFuncs.retainobject = nFuncs->retainobject;

NPNFuncs.releaseobject = nFuncs->releaseobject;

NPNFuncs.invoke = nFuncs->invoke;

NPNFuncs.invokeDefault = nFuncs->invokeDefault;

NPNFuncs.evaluate = nFuncs->evaluate;

NPNFuncs.getproperty = nFuncs->getproperty;

NPNFuncs.setproperty = nFuncs->setproperty;

NPNFuncs.removeproperty = nFuncs->removeproperty;

NPNFuncs.hasproperty = nFuncs->hasproperty;

NPNFuncs.hasmethod = nFuncs->hasmethod;

NPNFuncs.releasevariantvalue = nFuncs->releasevariantvalue;

NPNFuncs.setexception = nFuncs->setexception;

}

### NP\_Shutdown

The browser calls NP\_Shutdown when the last instance of the plug-in is destroyed. This is the place to release common plugin resources (i.e. the ones shared between all instances of the plugin). For example, if you allocated some resources in the NP\_Initialize, NP\_Shutdown is probably going to be the place where you should release them.

Since in our example we did not allocate any resources in the NP\_Initialize, our NP\_Shutdown implementation is going to be very simple :

NPError OSCALL NP\_Shutdown()

{

return NPERR\_NO\_ERROR;

}

## The Example Plugin

Plugin that will be described in this document will expose an object and a type to the page.

The object exposed will be called pluginObject and will have one method called ‘*method’* and one attribute called ‘*attribute’*. The user will be able to use it from JavaScript like this :

<SCRIPT>

Function example()

{

pluginObject.attribute = 500;

var temp = pluginObject.attribute;

pluginObject.method();

}

</SCRIPT>

Internally, pluginObject will be implemented as an object of class MyObject, which will be described later in the document.

Our plugin will also provide the user with an option of creating new objects of the same type as pluginObject.

<SCRIPT>

Function example2()

{

var temp = new pluginType();

temp.method();

temp.attribute = 400;

}

</SCRIPT>

This functionality will be implemented using an object of a class called MyType, which will be described later in the document.

## Implementing MyPlugin class

In our example, MyPlugin will be the place where we implement NPPluginFuncs. We will also store our custom NPObjects in this class.

In the plugin example, we will have all NPP functions, except NPP\_New and NPP\_Destroy, empty.

NPError MyPlugin::NPP\_SetWindow(NPP instance, NPWindow\* window)

{

return NPERR\_NO\_ERROR;

}

NPError MyPlugin::NPP\_NewStream(NPP instance, NPMIMEType type,

NPStream\* stream, NPBool seekable, uint16\_t\* stype)

{

return NPERR\_NO\_ERROR;

}

NPError MyPlugin::NPP\_DestroyStream(NPP instance, NPStream\* stream,

NPReason reason)

{

return NPERR\_NO\_ERROR;

}

…

The MyPlugin class will also contain NPNetscapeFuncs member that will get initialized during NP\_Initialize call.

### Creating a plugin instance

When browser comes across embed tag with a MIME type that was registered by a plugin, it will create a plugin instance and invoke NPP\_New on a plugin. The MIME type that was found on the page will be passed in as a parameter to NPP\_New so if a plugin has several MIME types registered, NPP\_New is the place to decide on a MIME type specific behaviour.

Our example plugin NPP\_New function will look like this :

NPError MyPlugin::NPP\_New(NPMIMEType pluginType,

NPP instance,

uint16\_t mode,

int16\_t argc,

char\* argn[],

char\* argv[],

NPSavedData\* saved)

{

if(instance == NULL)

return NPERR\_INVALID\_INSTANCE\_ERROR;

MyPlugin\* pPluginObj = new MyPlugin(instance);

if (pPluginObj == NULL)

return NPERR\_OUT\_OF\_MEMORY\_ERROR;

instance->pdata = pPluginObj;

return NPERR\_NO\_ERROR;

}

In NPP\_New we create a MyPlugin object which we will use to expose our custom plugin type and plugin object on the page. We will use a reference to the MyPlugin object as our

NPP->pdata .

In the constructor for the MyPlugin we will store the NPP structure for this plugin instance and create NPObjects for our custom object and our custom type.

MyPlugin:: MyPlugin (NPP instance)

{

mInstance=instance;

NPVariant temp;

NPObject \*sWindowObj;

NPN\_GetValue(mInstance, NPNVWindowNPObject, &sWindowObj);

mObject = (MyObject \*)NPN\_CreateObject(mInstance, &MyObject::\_npclass);

if (!mObject){…} //error handling

OBJECT\_TO\_NPVARIANT(mObject, temp);

NPIdentifier ObjectName = NPN\_GetStringIdentifier("pluginObject");

NPN\_SetProperty(mInstance, sWindowObj, myName,&temp);

mType = (MyType \*)NPN\_CreateObject(mInstance, & MyType::\_npclass);

if (!mType){…} //error handling

OBJECT\_TO\_NPVARIANT(mType, temp);

NPIdentifier ObjectName = NPN\_GetStringIdentifier("pluginType");

NPN\_SetProperty(mInstance, sWindowObj, myName,&temp);

NPN\_ReleaseObject(sWindowObj);

}

We created an NPObject, retrieved the browser window NPObject, and exposed the newly created NPObject to the browser window. As a result of this, the user will be able to communicate with our custom objects like this:

<SCRIPT>

function getAttribute()

{

var temp = pluginObject.attribute;

}

function createObject()

{

var temp = new pluginType();

}

</SCRIPT>

After calling NPN\_GetValue, browser will increment reference count for the returned object (NPNVWindowNPObject), so we have to release it once we are done with it, hence a call to NPN\_ReleaseObject(sWindowObj).

The other thing to keep in mind is the reference count for NPObjects we created. A call to NPN\_CreateObject will automatically set the reference count for mObject to 1. Since we are storing the reference to mObject in the MyPlugin instance as a class member, this is ok. However, if a reference to mObject was not kept within the plugin, then there would be a need for a NPN\_ReleaseObject(mObject) call after creation.

MyObject and MyPlugin constructor look almost identical. We will create all method and attribute name identifiers in the MyObject and MyPlugin constructor which will save us conversion of received identifiers to a string later on.

MyObject::MyObject(NPP npp)

{

mInstance=npp;

// methods

smethod\_id = NPN\_GetStringIdentifier("method");

// attributes

sattribute\_id = NPN\_GetStringIdentifier("attribute");

}

### Destroying a plugin instance

The browser will call NPP\_Destroy on a plugin when it needs to delete the plugin instance. This will usually be a result of navigating away from the page that had the plugin instance, closing the web page window or closing the browser. In our example, this will be the point when we release all the NPObjects we created and stored at plugin instance initialization.

NPError MyPlugin::NPP\_Destroy(NPP instance, NPSavedData\*\* save)

{

delete (MyPlugin\*)instance->pdata;

return NPERR\_NO\_ERROR;

}

MyPlugin::~MyPlugin()

{

NPN\_ReleaseObject(mObject);

NPN\_ReleaseObject(mType);

}

## Implementing MyObject and MyType class

Both MyObject and MyType will inherit from NPObject, so we will have to fill in their NPClass first:

NPClass MyObject::\_npclass = {

NP\_CLASS\_STRUCT\_VERSION,

MyObject::Allocate,

MyObject::\_Deallocate,

MyObject::\_Invalidate,

MyObject::\_HasMethod,

MyObject::\_Invoke,

MyObject::\_InvokeDefault,

MyObject::\_HasProperty,

MyObject::\_GetProperty,

MyObject::\_SetProperty,

MyObject::\_RemoveProperty,

MyObject::\_Enumerate,

MyObject::\_Construct

};

Since we need to have the NPClass structure available before we create a corresponding NPObject, we will have our NPClass structure point to static functions. We will then reroute those static functions to instance functions :

bool MyObject::\_HasMethod(NPObject \*npobj, NPIdentifier name)

{

return ((MyObject\*)npobj)->HasMethod(name);

}

The only exception is the Allocate functions which will create an instance of MyObject or MyType. This is the function that will be invoked as a result of NPN\_CreateObject.

NPObject\* MyObject::Allocate(NPP npp, NPClass \*aClass)

{

return (NPObject \*)new MyObject(npp);

}

We will delete the created instance in a call to \_Deallocate. This function will be invoked by the browser once the reference count for an NPObject goes down to 0.

void MyObject::\_Deallocate(NPObject \*npobj)

{

((MyObject\*)npobj)->Deallocate();

delete ((MyObject\*)npobj);

}

### Working with attributes and methods of MyObject

When a browser comes across JavaScript code that says

pluginObject.attribute = 500;

it will first invoke NPP\_HasProperty and NPP\_HasMethod to check if the NPObject in question had a method or property with name ‘attribute’. The implementation of those methods in our MyObject class will look something like this:

bool MyObject::HasMethod(NPIdentifier name)

{

return (name == smethod\_id);

}

bool MyObject::HasProperty(NPIdentifier name)

{

return (name == sattribute\_id);

}

Smethod\_id and sattribute\_id are the NPIdentifiers we created when we constructed the MyObject class (see [Creating a plugin instance](#_Creating_a_plugin)). This saves us converting received NPIdentifier to string for each call to NPP\_HasMethod and NPP\_HasProperty. It will also come in handy in other NPP functions, as you will see later.

It is not possible to have attributes and methods with the same name in one NPObjects. If a browser gets true as a response to NPP\_HasProperty (which is called first in the browser used by the author), it will not proceed to invoke NPP\_HasMethod.

If NPP\_HasProperty has returned true, the browser will invoke NPP\_SetProperty or NPP\_GetProperty as appropriate. In our example above, it will invoke NPP\_SetProperty which will then end in MyObject::\_setProperty which will look something like this:

bool MyJSObject::SetProperty(NPIdentifier name, const NPVariant \*value)

{

if (name == sattribute\_id) //check that this is the attribute to be set

{

if (NPVARIANT\_IS\_DOUBLE(\*value)) // check that the type matches

{

// invoke function that will handle setting of this property

setAttributeValue(NPVARIANT\_TO\_DOUBLE(\*value);

return true;

}

}

//property not found

return false;

}

*NPIdentifier name* is the identifier the browser found on the exposed object, and the *NPVariant \*value* is the new value of the property (500 in the example above). Again, we are using the preset NPIdentifiers to check which attribute to set.

MyObject::GetProperty would have similar implementation.

Special care should be taken when dealing with NPVariants that are strings. When returning a string back to browser (i.e. in NPN\_GetProperty), memory for the string should be allocated using NPN\_MemAlloc.

In case NPP\_HasMethod has returned true, the next function the browser will call is NPP\_Invoke, which will end in *MyObject::Invoke(NPIdentifier name, const NPVariant \*args, uint32\_t argCount, NPVariant \*result).* The implementation of MyObject::Invoke will be very similar to MyObject::SetProperty: we would check which method is being invoked by checking the *NPIdentifier name* passed in, we would retrieve the arguments passed in as *NPVariant\*args*, invoke desired method and return result in *NPVariant \*result*.

### Creating new objects with MyType class

When a browser comes across following JavaScript

var temp = new pluginType();

It will invoke NPP\_Construct on the pluginType NPObject which will, in our example, end in *MyType:\_NPConstruct (const NPVariant \*args, uint32\_t argCount, NPVariant \*result)*. If there were any arguments to our JavaScript constructor, they would be passed in as *NPVariant \*args.* The resulting object will be returned to browser through *NPVariant \*result.*

In our plugin example, MyType will be creating NPObjects of MyObject type so the implementation will look something like this:

bool MyType::Construct(const NPVariant \*args, uint32\_t argCount, NPVariant \*result)

{

MyObject\* temp =(MyObject\*) NPN\_CreateObject(mInstance, MyObject::\_npclass);

if (!temp)

{

return false;

}

OBJECT\_TO\_NPVARIANT(ngenericObj, \*result);

// NPN\_RetainObject(temp); //we are passing the new object to browser

// NPN\_ReleaseObject(temp); //we are not keeping a reference to temp within plugin

return true;

}

We have created the new object using a call to NPN\_CreateObject even though we have access to the MyObject class from our plugin. This is to allow the browser to do the memory management for the new object.

The commented out lines are there to make it more clear what is going on with the reference count for MyObject \*temp. After a call to NPN\_CreateObject, initial reference count for MyObject \*temp will be 1. We are not keeping the reference within our plugin, so we should release the temp object. However, we are returning the object to the browser which will also release it once it no longer needs it, so we should retain the object before giving it to the browser. We have skipped calling either of the functions as the initial reference count for \*temp is in this case correct.

# NPAPI reference

This chapter contains the description of mentioned NPAPI functions as they were at the time this document was created. The purpose of this part of the document is for easy reference during reading of this document. The description was retrieved from <https://developer.mozilla.org>

## NPN\_MemAlloc

### Summary

Allocates memory from the browser's memory space.

### Syntax

#include <npapi.h>

void \*NPN\_MemAlloc (uint32 size);

### Parameters

The function has the following parameters:

size

Size of memory, in bytes, to allocate in the browser's memory space.

### Returns

* If successful, the function returns a pointer to the allocated memory, in bytes.
* If insufficient memory is available, the plug-in returns null.

### Description

The plug-in calls NPN\_MemAlloc to allocate a specified amount of memory in the browser's memory space. If you allocate saved instance data with NPP\_Destroy, be sure to use NPN\_MemAlloc to allocate memory. This ensures that the browser can free the saved data at a later time with the equivalent of NPN\_MemFree.

Since the browser and plug-ins share the same memory space, NPN\_MemAlloc allows plug-ins to take advantage of any customized memory allocation scheme the application may have, and allows the application to manage its memory more flexibly and efficiently.

#### Mac OS

NPN\_MemAlloc is particularly important on Mac OS, since the Mac OS version of the browser frequently fills its memory partition with cached data that is only purged as necessary. Since NPN\_MemAlloc automatically frees cached information if necessary to fulfill the request, calls to NPN\_MemAlloc may succeed where direct calls to NewPtr fail.

Existing calls to NPN\_MemFlush have no effect. You only need to use NPN\_MemFlush in situations where you cannot use NPN\_MemAlloc, for example, when calling system methods that allocate memory indirectly.

## NPN\_MemFree

### Summary

Deallocates a block of allocated memory.

### Syntax

#include <npapi.h>

void NPN\_MemFree (void\* ptr);

### Parameters

The function has the following parameters:

ptr

Block of memory previously allocated using NPN\_MemAlloc.

### Description

NPN\_MemFree deallocates a block of memory that was allocated using NPN\_MemAlloc only. NPN\_MemFree does not free memory allocated by any other means.

## NPN\_GetValue

### Summary

Allows the plug-in to query the browser for information

### Syntax

#include <npapi.h>

NPError NPN\_GetValue(NPP instance,

NPNVariable variable,

void \*value);

### Parameters

This function has the following parameters:

instance

Pointer to the current plug-in instance.

variable

Information the call gets. Values for NPNVariable:

* NPNVxDisplay =1: Unix only: Returns the current Display
* NPNVxtAppContext: Unix only: Returns the application's XtAppContext
* NPNVnetscapeWindow: MS Windows and Unix/X11 only:
  + MS Windows: Gets the native window on which plug-in drawing occurs; returns HWND
  + Unix/X11: Gets the browser toplevel window in which the plug-in is displayed; returns Window
* NPNVjavascriptEnabledBool: Tells whether JavaScript is enabled; true=JavaScript enabled, false=not enabled
* NPNVasdEnabledBool: Tells whether SmartUpdate (former name: ASD) is enabled; true=SmartUpdate enabled, false=not enabled
* NPNVisOfflineBool: Tells whether offline mode is enabled; true=offline mode enabled, false=not enabled
* NPNVToolkit:
* NPNVSupportsXEmbedBool:
* NPNVWindowNPObject: Returns the NPObject\* pointer for the DOM window object
* NPNVPluginElementNPObject:
* NPNVSupportsWindowless: Tells whether the browser supports windowless plugins. The value parameter should point to a NPBool, which will be set appropriately if the function returns NPERR\_NO\_ERROR.
* NPNVprivateModeBool: Indicates whether or not the browser is currently in private browsing mode.  The value parameter should point to an NPBool, which will be set appropriately if the function returns NPERR\_NO\_ERROR.
* NPNVpluginDrawingModel: (Mac only)
* NPNVsupportsQuickDrawBool : (Mac only)
* NPNVsupportsCoreGraphicsBool : (Mac only)

value

Function returns the name of the plug-in in the value parameter.

### Returns

* If successful, the function returns NPERR\_NO\_ERROR.
* If unsuccessful, the plug-in is not loaded and the function returns an error code. For possible values, see Error Codes.

### Description

NPN\_GetValue returns the browser information set with NPN\_SetValue. The queried information is returned in the value parameter.

#### Unix

The platform-specific values for this parameter are NPNVxDisplay (the current Display), NPNVxtAppContext (the browser's XtAppContext), and NPNVnetscapeWindow (the browser toplevel window in which the plugin is displayed).

You can use NPNVnetscapeWindow to obtain the appropriate Window for the WM\_TRANSIENT\_FOR property of a dialog box created from the plug-in.

#### Microsoft Windows

You can use this method to help create a menu or dialog box for a windowless plug-in. In order to bring up popup menus and modal dialogs, a plug-in needs a parent window. A windowless plug-in does not receive its own native window. Instead, it draws directly into the drawable given to it. Use the NPNVnetscapeWindow value to get the native window on which plug-in drawing occurs.

The method returns a value of type HWND. In many cases, a plug-in may still have to create its own window (a transparent child window of the browser window) to act as the owner window for popup menus and modal dialogs. This transparent child window can have its own WindowProc within which the plug-in can deal with WM\_COMMAND messages sent to it a result of tracking the popup menu or modal dialog.

## NPN\_CreateObject

### Summary

Allocates a new NPObject.

### Syntax

#include <npruntime.h>

NPObject \*NPN\_CreateObject(NPP npp, NPClass \*aClass);

### Parameters

The function has the following parameters:

npp

The NPP indicating which plugin wants to instantiate the object.

aClass

The class to instantiate an object of.

### Returns

The newly-allocated NPObject.

### Description

If the given NPClass provides an allocate function it is used to allocate the storage for the object and the NPP argument passed to NPN\_CreateObject() is passed along to that function. If no allocate function is provided, malloc() is called to allocate enough memory to hold an NPObject. The newly created NPObject's reference count is initialized to 1 before it is returned.

## NPN\_RetainObject

### Summary

Increments the reference count of the given NPObject.

### Syntax

#include <npruntime.h>

NPObject \*NPN\_RetainObject(NPObject \*npobj);

### Parameters

The function has the following parameter:

npobj

The NPObject to retain.

### Returns

A pointer back to the same object.

## NPN\_ReleaseObject

### Summary

Decrements the reference count of the given NPObject. If the reference count reaches 0, the NPObject is deallocated by calling its deallocate function if one is provided; if one is not provided, free() is used.

### Syntax

#include <npruntime.h>

void NPN\_ReleaseObject(NPObject \*npobj);

### Parameters

The function has the following parameter:

npobj

The NPObject whose reference count should be decremented.

### Returns

Nothing.

## NPN\_Invoke

### Syntax

#include <npruntime.h>

bool NPN\_Invoke( NPP npp,

NPObject \*npobj,

NPIdentifier methodName,

const NPVariant \*args,

uint32\_t argCount,

NPVariant \*result);

### Parameters

The function has the following parameters:

npp

The NPP indicating which plugin wants to call the method on the object.

npobj

The object on which to invoke a method.

methodName

A string identifier indicating the name of the method to invoke.

args

An array of arguments to pass to the method.

argCount

The number of arguments in the args array.

result

A pointer to an NPVariant to receive the result returned by the method.

### Returns

true if the method was successfully invoked, otherwise false. If the method was invoked successfully, any return value is stored in the NPVariant specified by result.

When the caller no longer needs the result, it must call NPN\_ReleaseVariantValue() to release it.

### Description

The method arguments are passed as an array of NPVariants, and the number of arguments is passed in. The result of the method invocation is returned through an NPVariant result parameter. If the method invocation succeeds, NPN\_Invoke() returns true, else false.

**Note:** Early on in the development of the scriptability API this method used to be called NPN\_Call() but was renamed for consistency.

## NPN\_InvokeDefault

### Summary

Invokes the default method, if one exists, on the given NPObject.

### Syntax

#include <npruntime.h>

bool NPN\_InvokeDefault( NPP npp,

NPObject \*npobj,

const NPVariant \*args,

uint32\_t argCount,

NPVariant \*result);

### Parameters

The function has the following parameters:

npp

The NPP indicating which plugin wants to call the default method on the object.

npobj

The object on which to invoke the default method.

args

An array of arguments to pass to the default method.

argCount

The number of arguments in the args array.

result

A pointer to an NPVariant to receive the result returned by the default method.

### Returns

true if the default method was successfully invoked, otherwise false. If the default method was invoked successfully, any return value is stored in the NPVariant specified by result.

When the caller no longer needs the result, it must call NPN\_ReleaseVariantValue() to release it.

### Description

The method arguments are passed as an array of NPVariants, and the number of arguments is passed in. The result of the method invocation is returned through an NPVariant result parameter. If the method invocation succeeds, NPN\_Invoke() returns true, else false.

## NPN\_Evaluate

### Summary

Evaluates a script in the scope of the specified NPObject.

### Syntax

#include <npruntime.h>

bool NPN\_Evaluate( NPP npp,

NPObject \*npobj,

NPString \*script,

NPVariant \*result);

### Parameters

The function has the following parameters:

npp

The NPP indicating which plugin instance's window to evaluate the script in.

npobj

The scope object.

script

The script to evaluate.

result

On return, contains the value returned by the script.

### Returns

true if the script was evaluated successfully, otherwise false.

### Description

The script is evaluated in the context of the window that the calling plugin instance (the npp argument) is loaded in.

**Note:** The caller must call NPN\_ReleaseVariantValue() to release the returned value when it's no longer needed.

## NPN\_GetProperty

### Summary

Gets the value of a property on the specified NPObject.

### Syntax

#include <npruntime.h>

bool NPN\_GetProperty( NPP npp,

NPObject \*npobj,

NPIdentifier propertyName,

NPVariant \*result);

### Parameters

The function has the following parameters:

npp

The NPP indicating which plugin instance's is making the request.

npobj

The object from which a value is to be retrieved.

propertyName

A string identifier indicating the name of the property whose value is to be retrieved.

result

On return, contains the value of the specified property.

### Returns

true if the value was retrieved successfully, otherwise false.

**Note:** The caller must call NPN\_ReleaseVariantValue() to release the returned value when it's no longer needed.

## NPN\_SetProperty

### Summary

Sets the value of a property on the specified NPObject.

### Syntax

#include <npruntime.h>

bool NPN\_SetProperty( NPP npp,

NPObject \*npobj,

NPIdentifier propertyName,

const NPVariant \*value);

### Parameters

The function has the following parameters:

npp

The NPP indicating which plugin instance's is making the request.

npobj

The object on which a value is to be set.

propertyName

A string identifier indicating the name of the property whose value is to be set.

value

The value to store in the specified property.

### Returns

true if the valu

e was set successfully, otherwise false.

## NPN\_RemoveProperty

### Syntax

#include <npruntime.h>

bool NPN\_RemoveProperty( NPP npp,

NPObject \*npobj,

NPIdentifier propertyName);

### Parameters

The function has the following parameters:

npp

The NPP indicating which plugin instance is making the request.

npobj

The object on which a property is to be deleted.

propertyName

A string identifier indicating the name of the property to remove.

### Returns

true if the property was removed successfully, otherwise false.

## NPN\_HasProperty

### Summary

Determines whether or not the specified NPObject has a particular property.

### Syntax

#include <npruntime.h>

bool NPN\_HasProperty( NPP npp,

NPObject \*npobj,

NPIdentifier propertyName);

### Parameters

The function has the following parameters:

npp

The NPP indicating which plugin instance is making the request.

npobj

The object on which to look for the property.

propertyName

A string identifier indicating the name of the property to look for.

### Returns

true if the property exists on the specified object, otherwise false.

## NPN\_HasMethod

### Summary

Determines whether or not the specified NPObject has a particular method.

### Syntax

#include <npruntime.h>

bool NPN\_HasMethod( NPP npp,

NPObject \*npobj,

NPIdentifier methodName);

### Parameters

The function has the following parameters:

npp

The NPP indicating which plugin instance is making the request.

npobj

The object on which to look for the method.

methodName

A string identifier indicating the name of the method to look for.

### Returns

true if the method exists on the specified object, otherwise false.

## NPN\_SetException

### Summary

A plugin can call this function to indicate that a call to one of the plugin's NPObjects generated an error.

### Syntax

#include <npruntime.h>

void NPN\_SetException( NPObject \*npobj,

const NPUTF8 \*message);

### Parameters

The function has the following parameters:

npobj

The object on which the exception occurred.

message

A UTF-8 string describing the error that occurred.

### Returns

Nothing.

## NPN\_ReleaseVariantValue

### Summary

NPN\_ReleaseVariantValue() releases the value in the given variant.

### Syntax

#include <npruntime.h>

void NPN\_ReleaseVariantValue(NPVariant \*variant);

### Parameters

The function has the following parameters:

variant

The variant whose value is to be released.

### Returns

Nothing.

### Description

NPN\_ReleaseVariantValue() releases the value in the given variant. This must always be called on result variants and such in this API, i.e. any NPVariant whose value comes from a call that passes back an NPVariant must be released using this function. Access to the value in an NPVariant that has been released will result in undefined behavior.

NPN\_ReleaseVariantValue() will call NPN\_ReleaseObject() on NPVariants of type NPVARIANTTYPE\_OBJECT, and NPN\_FreeMem() on NPVariants of type NPVARIANTTYPE\_STRING.

## NPN\_GetStringIdentifier

### Summary

Returns an opaque identifier for the string that is passed in.

### Syntax

#include <npruntime.h>

NPIdentifier NPN\_GetStringIdentifier(const NPUTF8 \*name);

### Parameters

The function has the following parameters:

name

The string for which an opaque identifier should be returned.

### Returns

An NPIdentifier identifying the string.

### Description

All calls for the same string are guaranteed to return the same exact identifier.

## NPN\_GetStringIdentifiers

### Summary

Returns an array of opaque identifiers for the names that are passed in.

### Syntax

#include <npruntime.h>

void NPN\_GetStringIdentifiers( const NPUTF8 \*\*names,

int32\_t nameCount,

NPIdentifier \*identifiers);

### Parameters

The function has the following parameters:

names

An array of strings for which opaque identifiers should be returned.

nameCount

The number of strings in the names array.

identifiers

A pointer to the first element of an array to fill with the identifiers for the strings specified by names.

### Returns

On return, the identifiers array is filled out with the identifiers for each of the strings in the names array.

### Description

As is the case with NPN\_GetStringIdentifier(), all calls for the same strings are guaranteed to return the same exact identifiers.

## NPN\_GetIntIdentifier

### Summary

Returns an opaque identifier for the integer that is passed in.

### Syntax

#include <npruntime.h>

NPIdentifier NPN\_GetIntIdentifier(int32\_t intid);

### Parameters

The function has the following parameter:

intid

The integer for which an opaque identifier should be returned.

### Returns

An NPIdentifier identifying the integer.

### Description

All calls for the same integer are guaranteed to return the same exact identifier.

## NPN\_IdentifierIsString

### Summary

Determines whether or not an identifier is a string.

### Syntax

#include <npruntime.h>

bool NPN\_IdentifierIsString(NPIdentifier identifier);

### Parameters

The function has the following parameter:

identifier

The identifier whose type is to be examined.

### Returns

true if the identifier is a string identifier, or false otherwise.

## NPN\_UTF8FromIdentifier

### Summary

Returns the UTF-8 string corresponding to the given string identifier.

### Syntax

#include <npruntime.h>

NPUTF8 \*NPN\_UTF8FromIdentifier(NPIdentifier identifier);

### Parameters

The function has the following parameter:

identifier

The string identifier whose corresponding string should be returned.

### Returns

A UTF-8 string as a sequence of NPUTF8 bytes, or NULL if the specified identifier isn't a string identifier.

### Description

Once the caller is done with the returned string, the caller is responsible for deallocating the memory used by the string by calling NPN\_MemFree().

## NPN\_IntFromIdentifier

### Summary

Returns the integer value corresponding to the given integer identifier.

### Syntax

#include <npruntime.h>

int32\_t NPN\_IntFromIdentifier(NPIdentifier identifier);

### Parameters

The function has the following parameter:

identifier

The integer identifier whose corresponding integer value should be returned.

### Returns

An integer corresponding to the specified identifier.

**Note:** The behaviour of this function is undefined if identifier isn't an integer identifier.

## NPP\_New

### Summary

Creates a new instance of a plug-in.

### Syntax

#include <npapi.h>

NPError NPP\_New(NPMIMEType pluginType,

NPP instance, uint16 mode,

int16 argc, char \*argn[],

char \*argv[], NPSavedData \*saved);

### Parameters

The function has the following parameters:

pluginType

Pointer to the MIME type for new plug-in instance.

instance

Contains instance-specific private data for the plug-in and the browser. This data is stored in instance->pdata.

mode

Display mode of plug-in. Values:

* NP\_EMBED: (1) Instance was created by an EMBED tag and shares the browser window with other content.
* NP\_FULL: (2) Instance was created by a separate file and is the primary content in the window.

argc

Number of HTML arguments in the EMBED tag for an embedded plug-in; determines the number of attributes in the argn and argv arrays.

argn[]

Array of attribute names passed to the plug-in from the EMBED tag.

argv[]

Array of attribute values passed to the plug-in from the EMBED tag.

saved

Pointer to data saved by NPP\_Destroy for a previous instance of this plug-in at the same URL. If non-null, the browser passes ownership of the NPSavedData object back to the plug-in. The plug-in is responsible for freeing the memory for the NPSavedData and the buffer it contains.

### Returns

* If successful, the function returns NPERR\_NO\_ERROR.
* If unsuccessful, the function returns an error code. For possible values, see [Error Codes](#_Error_Codes).

### Description

NPP\_New creates a new instance of a plug-in. It is called after NP\_Initialize and is passed the MIME type, embedded or full-screen display mode, and, for embedded plug-ins, information about HTML EMBED arguments.

A new instance pointer (NPP instance) is created by the plug-in, which is valid until the instance is destroyed with NPP\_Destroy.

If instance data was saved from a previous instance of the plug-in by the NPP\_Destroy function, it is returned in the saved parameter for the current instance to use.

All attributes in the EMBED tag (standard and private) are passed in NPP\_New in the argn and argv arrays. The browser ignores any non-standard attributes within an EMBED tag. This gives developers a chance to use private attributes to communicate instance-specific options or other information to the plug-in. Place private options at the end of the list of standard attributes in the EMBED Tag.

## NPP\_Destroy

### Summary

Deletes a specific instance of a plug-in.

### Syntax

#include <npapi.h>

NPError NPP\_Destroy(NPP instance,

NPSavedData \*\*save);

### Parameters

The function has the following parameters:

instance

Pointer to the plug-in instance to delete.

\*\*save

State or other information to save for reuse by a new instance of this plug-in at the same URL. Passed to NPP\_New.

### Returns

* If successful, the function returns NPERR\_NO\_ERROR.
* If unsuccessful, the plug-in is not loaded and the function returns an error code. For possible values, see [Error Codes](#_Error_Codes).

### Description

NPP\_Destroy releases the instance data and resources associated with a plug-in. The browser calls this function when a plug-in instance is deleted, typically because the user has left the page containing the instance, closed the window, or quit the browser. You should delete any private instance-specific information stored in the plug-in's instance->pdata at this time.

If this function is deleting the last instance of a plug-in, NP\_Shutdown is subsequently called. Use NP\_Shutdown to delete any data allocated in NP\_Initialize and intended to be shared by all instances of a plug-in.

Use the optional save parameter if you want to save and reuse some state or other information. Upon the user's return to the page, this information is passed to the new plug-in instance when it is created with NPP\_New.

Avoid trying to save critical data with this function. Ownership of the buf field of the NPSavedData structure passes from the plug-in to the browser when NPP\_Destroy returns. The browser can and will discard this data based on arbitrary criteria such as its size and the user's page history.

To ensure that the browser does not crash or leak memory when the saved data is discarded, NPSavedData's buf field should be a flat structure (a simple structure with no allocated substructures) allocated with NPN\_MemAlloc.

##### Mac OS

If you want to restore state information if this plug-in is later recreated, use NP\_MemAlloc to create an NPSavedData structure.

NOTE: You should not perform any graphics operations in NPP\_Destroy as the instance's window is no longer guaranteed to be valid.

## NPP\_SetWindow

### Summary

Tells the plug-in when a window is created, moved, sized, or destroyed.

### Syntax

#include <npapi.h>

NPError NPP\_SetWindow(NPP instance, NPWindow \*window);

### Parameters

The function has the following parameters:

instance

Pointer to the current plug-in instance. Must be embedded or full-screen.

window

Pointer to the window into which the instance draws. The window structure contains a window handle and values for top left corner, width, height, and clipping rectangle (see note on Unix below).

### Returns

* If successful, the function returns NPERR\_NO\_ERROR.
* If unsuccessful, the plug-in is not loaded and the function returns an error code. For possible values, see Error Codes.

### Description

The browser calls NPP\_SetWindow after creating the instance to allow drawing to begin. Subsequent calls to NPP\_SetWindow indicate changes in size or position; these calls pass the same NPWindow object each time, but with different values. If the window handle is set to null, the window is destroyed. In this case, the plug-in must not perform any additional graphics operations on the window and should free any associated resources.

The data structure passed in NPP\_SetWindow is an NPWindow object, which contains the coordinates of the instance's area and various platform-specific data. This window is valid for the life of the instance, or until NPP\_SetWindow is called again with a different value.

For windowed plug-ins on Windows and Unix, the window parameter contains a handle to a subwindow of the browser window hierarchy. On Mac OS, this field points to an NP\_Port structure. For windowless plug-ins, it is a platform-specific handle to a drawable.

Before setting the window parameter to point to a new window, it is a good idea to compare the information about the new window to the previous window (if one existed) to account for any changes.

NOTE: NPP\_SetWindow is useful only for embedded (NP\_EMBED) or full-screen (NP\_FULL) plug-ins, which are drawn into windows. It is irrelevant for hidden plug-ins.

## NPP\_NewStream

### Summary

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

### Syntax

#include <npapi.h>

NPError NPP\_NewStream(NPP instance,

NPMIMEType type,

NPStream\* stream,

NPBool seekable,

uint16\* stype);

### Parameters

The function has the following parameters:

instance

Pointer to current plug-in instance.

type

Pointer to MIME type of the stream.

stream

Pointer to new stream.

seekable

Boolean indicating whether the stream is seekable:

* true: Seekable. Stream supports random access through calls to NPN\_RequestRead (for example, local files or HTTP servers that support byte-range requests).
* false: Not seekable. The browser must copy data in the stream to the local cache to satisfy random access requests made through NPN\_RequestRead.

stype

Out parameter. The plugin should set this value to request a mode for the stream. For more information about each of these values, see Directions in this section.

* **NP\_NORMAL** (Default): Delivers stream data to the instance in a series of calls to NPP\_WriteReady and NPP\_Write.
* **NP\_ASFILEONLY**: Saves stream data to a file in the local cache.
* **NP\_ASFILE**: File download. Like NP\_ASFILEONLY except that data is delivered to the plug-in as it is saved to the file (as in mode NP\_NORMAL).
* **NP\_SEEK**: Stream data randomly accessible by the plug-in as needed, through calls to NPN\_RequestRead.

Implementation note: some plugins, notably Silverlight, do not set this outparam, and rely on the outparam being initialized to a default NP\_NORMAL value.

### Returns

* If successful, the function should return NPERR\_NO\_ERROR.
* If unsuccessful, the function should return one of the NPError Error Codes.  This will cause the browser to destroy the stream without calling NPP\_DestroyStream.

### Description

NPP\_NewStream notifies the plug-in when a new stream is created. The NPStream\* pointer is valid until the stream is destroyed. The plug-in can store plug-in-private data associated with the stream in stream->pdata. The MIME type of the stream is provided by the type parameter.

The data in the stream can be the file specified in the SRC attribute of the EMBED tag, for an embedded instance, or the file itself, for a full-page instance. A plug-in can also request a stream with the function NPN\_GetURL. The browser calls NPP\_DestroyStream when the stream completes (either successfully or abnormally). The plug-in can terminate the stream itself by calling NPN\_DestroyStream.

The parameter stype defines the mode of the stream. Values:

* NP\_NORMAL (Default): Delivers stream data to the instance in a series of calls to NPP\_WriteReady and NPP\_Write. The plug-in can process the data progressively as it arrives from the network or file system.
* NP\_ASFILEONLY: The browser saves stream data to a file in the local cache. When the stream is complete, the browser calls NPP\_StreamAsFile to deliver the path of the file to the plug-in. If the stream comes from a local file, the NPP\_Write and NPP\_WriteReady functions are not called. NPP\_StreamAsFile is simply called immediately. This mode allows the plug-in full random access to the data using platform-specific file operations.
* NP\_ASFILE: File download. Differs from NP\_ASFILEONLY in that data is delivered to the plug-in, through a series of calls to NPP\_WriteReady and NPP\_Write, as it is saved to the file (as in mode NP\_NORMAL). When the stream is complete, the browser calls NPP\_StreamAsFile to deliver the path of the file to the plug-in. If the data in the stream comes from a file that is already local, the data is read, sent to the plug-in through NPP\_Write, and written to a file in the local cache.

NOTE: Most plug-ins that need the stream saved to a file should use the more efficient mode NP\_ASFILEONLY (above); this mode is preserved for compatibility only.

* NP\_SEEK: Stream data is not automatically delivered to the instance, but can be randomly accessed by the plug-in as needed, through calls to NPN\_RequestRead. If the stream is not seekable, placing the stream in NP\_SEEK mode causes the browser to save the entire stream to the disk cache. NPN\_RequestRead requests are only fulfilled when all data has been read and stored in the cache. As an optimization to extract the maximum benefit from existing network connections, the browser continues to read data sequentially out of the stream (as in mode NP\_NORMAL) until the first NPN\_RequestRead call is made.

NOTE: In any mode other than NP\_SEEK, the application should call NPP\_DestroyStream once all data in the stream has been written to the plug-in. The plug-in can also request termination of the stream at any time by calling NPN\_DestroyStream.

## NPN\_DestroyStream

### Summary

Tells the plug-in that a stream is about to be closed or destroyed.

### Syntax

#include <npapi.h>

NPError NPP\_DestroyStream(NPP instance,

NPStream\* stream,

NPReason reason);

### Parameters

The function has the following parameters:

instance

Pointer to current plug-in instance.

stream

Pointer to current stream.

reason

Reason the stream was destroyed. Values:

* NPRES\_DONE (Most common): Completed normally; all data was sent to the instance.
* NPRES\_USER\_BREAK: User canceled stream directly by clicking the Stop button or indirectly by some action such as deleting the instance or initiating higher-priority network operations.
* NPRES\_NETWORK\_ERR: Stream failed due to problems with network, disk I/O, lack of memory, or other problems.

### Returns

* If successful, the plug-in should return NPERR\_NO\_ERROR.
* If unsuccessful, the plug-in should return an error code. For possible values, see Error Codes.

### Description

The browser calls the NPP\_DestroyStream function when a data stream sent to the plug-in is finished, either because it has completed successfully or terminated abnormally. After this, the browser deletes the NPStream object.

You should delete any private data allocated in stream->pdata at this time, and should not make any further references to the stream object.

## NPN\_StreamAsFile

### Summary

Provides a local file name for the data from a stream.

### Syntax

#include <npapi.h>

void NPP\_StreamAsFile(NPP instance,

NPStream\* stream,

const char\* fname);

### Parameters

The function has the following parameters:

instance

Pointer to current plug-in instance.

stream

Pointer to current stream.

fname

Pointer to full path to a local file. If an error occurs while retrieving the data or writing the file, fname may be null.

### Description

When the stream is complete, the browser calls NPP\_StreamAsFile to provide the instance with a full path name for a local file for the stream. NPP\_StreamAsFile is called for streams whose mode is set to NP\_ASFILEONLY or NP\_ASFILE only in a previous call to NPP\_NewStream.

If an error occurs while retrieving the data or writing the file, the file name (fname) is null.

## NPP\_WriteReady

### Summary

Determines maximum number of bytes that the plug-in can consume.

### Syntax

#include <npapi.h>

int32 NPP\_WriteReady(NPP instance, NPStream\* stream);

### Parameters

The function has the following parameters:

instance

Pointer to the current plug-in instance.

stream

Pointer to the current stream.

### Returns

Returns the maximum number of bytes that an instance is prepared to accept from the stream.

### Description

The browser calls NPP\_WriteReady before each call to NPP\_Write to determine whether a plug-in can receive data and how many bytes it can receive. This function allows the browser to send only as much data to the instance as it can handle at one time, making resource use more efficient for both the browser and plug-in.

The NPP\_Write function may pass a larger buffer, but the plug-in is required to consume only the amount of data returned by NPP\_WriteReady.

The browser can write a smaller amount of data if desired or necessary; for example, if only 8K of data is available in a network buffer. If the plug-in is allocating memory for the entire stream at once (an AS\_FILE stream), it can return a very large number. Because it is not processing streaming data, the browser can pass as much data to the instance as necessary in a single NPP\_Write.

If the browser receives a value of zero, the data flow temporarily stops. The browser checks to see if the plug-in can receive data again by resending the data at regular intervals.

## NPP\_Write

### Summary

Delivers data to a plug-in instance. (Remark: Hence the name "NPP\_Write" is misleading - just think of:"Data\_Arrived")

### Syntax

#include <npapi.h>

int32 NPP\_Write(NPP instance,

NPStream\* stream,

int32 offset,

int32 len,

void\* buf);

### Parameters

The function has the following parameters:

instance

Pointer to the current plug-in instance.

stream

Pointer to the current stream.

offset

Offset in bytes of buf from the beginning of the data in the stream. Can be used to check stream progress or by range requests from NPN\_RequestRead.

len

Length in bytes of buf; number of bytes accepted.

buf

Buffer of data, delivered by the stream, that contains len bytes of data offset bytes from the start of the stream. The buffer is allocated by the browser and is deleted after returning from the function, so the plug-in should make a copy of the data it needs to keep.

### Returns

* The plug-in should return the number of bytes written (consumed by the instance). If the return value is smaller than the size of the buffer, the browser sends the remaining data to the plug-in through subsequent calls to NPP\_WriteReady and NPP\_Write.
* If unsuccessful, the function signals an error by returning a negative value. The browser will destroy the stream by calling NPP\_DestroyStream.

### Description

The browser calls the NPP\_Write function to deliver the data specified in a previous NPP\_WriteReady call to the plug-in. A plug-in must consume at least as many bytes as indicated in the NPP\_WriteReady call.

After a stream is created by a call to NPP\_NewStream, the browser calls NPP\_Write either:

* If the plug-in requested a normal-mode stream, the data in the stream is delivered to the plug-in instance in a series of calls to NPP\_WriteReady and NPP\_Write.
* If the plug-in requested a seekable stream, the NPN\_RequestRead function requests reads of a specified byte range that results in a series of calls to NPP\_WriteReady and NPP\_Write.

The plug-in can use the offset parameter to track the bytes that are written. This gives you different information depending in the type of stream. In a normal-mode stream., the parameter value increases as the each buffer is written. The buf parameter is not persistent, so the plug-in must process data immediately or allocate memory and save a copy of it. In a seekable stream with byte range requests, you can use this parameter to track NPN\_RequestRead requests.

## NPP\_Print

### Summary

Requests a platform-specific print operation for an embedded or full-screen plug-in.

### Syntax

#include <npapi.h>

void NPP\_Print(NPP instance, NPPrint\* PrintInfo);

### Parameters

The function has the following parameters:

instance

Pointer to the current plug-in instance. Must be embedded or full-screen.

printInfo

Pointer to NPPrint structure.

### Description

NPP\_Print is called when the user requests printing for a web page that contains a visible plug-in (either embedded or full-page). It uses the print mode set in the NPPrint structure in its printInfo parameter to determine whether the plug-in should print as an embedded plug-in or as a full-page plug-in.

* An embedded plug-in shares printing with the browser; the plug-in prints the part of the page it occupies, and the browser handles everything else, including displaying print dialog boxes, getting the printer device context, and any other tasks involved in printing, as well as printing the rest of the page. For an embedded plug-in, set the printInfo field to NPEmbedPrint.
* A full-page plug-in handles all aspects of printing itself. For a full-page plug-in, set the printInfo field to NPFullPrint or null.

For information about printing on your platform, see your platform documentation.

#### MS Windows

On MS Windows printInfo->print.embedPrint.platformPrint is the device context (DC) handle. Be sure to cast this to type HDC.

The coordinates for the window rectangle are in TWIPS format. This means that you need to convert the x-y coordinates using the Windows API call DPtoLP when you output text.

## NPP\_HandleEvent

### Summary

Delivers a platform-specific window event to the instance.

* For Windowed Plug-ins: Currently used only on Mac OS.
* For Windowless Plug-ins: All platforms.

### Syntax

#include <npapi.h>

int16 NPP\_HandleEvent(NPP instance, void\* event);

### Parameters

The function has the following parameters:

instance

Pointer to the current plug-in instance.

event

Platform-specific value representing the event handled by the function. Values:

* MS Windows: Pointer to NPEvent structure
* Mac OS: Pointer to a standard Mac OS EventRecord
* Unix/X11: Pointer to a standard Xlib XEvent

For a list of possible events, see NPEvent.

### Returns

* If the plug-in handles the event, the function should return true.
* If the plug-in ignores the event, the function returns false.

### Description

The browser calls NPP\_HandleEvent to tell the plug-in when events take place in the plug-in's window or drawable area. The plug-in either handles or ignores the event, depending on the value given in the event parameter of this function. For a list of event types the application is responsible for delivering to the plug-in, see the NPEvent structure.

#### MS Windows

The browser gives each windowed plug-in its own native window, often a child window of the browser window, to draw into. The plug-in has complete control over drawing and event handling within that window.

#### Mac OS

The browser does not give a windowed plug-in a native window, because the Mac OS platform does not support child windows. Instead, the windowed plug-in draws into the graphics port associated with the the browser window, at the offset that the browser specifies. For this reason, NPP\_HandleEvent is only way the plug-in can receive events from its host application on Mac OS. When NPP\_HandleEvent is called, the current port is set up so that its origin matches the top-left corner of the plug-in. A plug-in does not need to set up the current port for mouse coordinate translation.

## NPP\_URLNotify

### Summary

Notifies the plug-in instance of the completion of a URL request.

### Syntax

#include <npapi.h>

void NPP\_URLNotify(NPP instance,

const char\* url,

NPReason reason,

void\* notifyData);

### Parameters

The function has the following parameters:

instance

Pointer to the current plug-in instance.

url

URL of the NPN\_GetURLNotify() or NPN\_PostURLNotify() request that has completed.

reason

Reason code for completion of request. Values:

* NPRES\_DONE (most common): Completed normally.
* NPRES\_USER\_BREAK: User canceled stream directly by clicking the Stop button or indirectly by some action such as deleting the instance or initiating higher-priority network operations.
* NPRES\_NETWORK\_ERR: Stream failed due to problems with network, disk I/O, lack of memory, or other problems.

notifyData

Plug-in-private value for associating a previous NPN\_GetURLNotify() or NPN\_PostURLNotify() request with a subsequent NPP\_URLNotify() call.

### Description

The browser calls NPP\_URLNotify() after the completion of a NPN\_GetURLNotify() or NPN\_PostURLNotify() request to inform the plug-in that the request was completed and supply a reason code for the completion.

The most common reason code is NPRES\_DONE, indicating simply that the request completed normally. Other possible reason codes are NPRES\_USER\_BREAK, indicating that the request was halted due to a user action (for example, clicking the Stop button), and NPRES\_NETWORK\_ERR, indicating that the request could not be completed, perhaps because the URL could not be found.

The parameter notifyData is the plug-in-private value passed as an argument by a previous NPN\_GetURLNotify() or NPN\_PostURLNotify() call, and can be used as an identifier for the request.

## NPP\_GetValue

### Summary

Allows the browser to query the plug-in for information.

### Syntax

#include <npapi.h>

NPError NPP\_GetValue(void \*instance,

NPPVariable variable,

void \*value);

### Parameters

The function has the following parameters:

instance

Pointer to the plugin instance from which the value should come.

variable

Variable for which the browser (caller) would like a value. The full list is in the NPAPI headers.

value

Value for the requested variable.

### Returns

* If successful, the function returns NPERR\_NO\_ERROR.
* If unsuccessful, the function returns an error code. For possible values, see Error Codes.

### Description

NPP\_GetValue retrieves instance variables.

## NPP\_SetValue

### Summary

Implemented by plugins. This call is used to inform plugins of variable information controlled by the browser.

### Syntax

#include <npapi.h>

NPError NPP\_SetValue(void \*instance,

NPNVariable variable,

void \*value);

### Parameters

The function has the following parameters:

instance

Pointer to plugin instance on which to set the variable.

variable

The variable being set.

value

Value for the variable being set.

### Returns

* If successful, the function should return NPERR\_NO\_ERROR.
* If unsuccessful, the function should return the most relevant NPAPI error code. For possible values, see Error Codes.

### Description

None

## Error Codes

|  |  |  |
| --- | --- | --- |
| Code | Value | Description |
| **NPERR\_NO\_ERROR** | 0 | No errors occurred |
| **NPERR\_GENERIC\_ERROR** | 1 | Error with no specific error code occurred |
| **NPERR\_INVALID\_INSTANCE\_ERROR** | 2 | Invalid instance passed to the plug-in |
| **NPERR\_INVALID\_FUNCTABLE\_ERROR** | 3 | Function table invalid |
| **NPERR\_MODULE\_LOAD\_FAILED\_ERROR** | 4 | Loading of plug-in failed |
| **NPERR\_OUT\_OF\_MEMORY\_ERROR** | 5 | Memory allocation failed |
| **NPERR\_INVALID\_PLUGIN\_ERROR** | 6 | Plug-in missing or invalid |
| **NPERR\_INVALID\_PLUGIN\_DIR\_ERROR** | 7 | Plug-in directory missing or invalid |
| **NPERR\_INCOMPATIBLE\_VERSION\_ERROR** | 8 | Versions of plug-in and Communicator do not match |
| **NPERR\_INVALID\_PARAM** | 9 | Parameter missing or invalid |
| **NPERR\_INVALID\_URL** | 10 | URL missing or invalid |
| **NPERR\_FILE\_NOT\_FOUND** | 11 | File missing or invalid |
| **NPERR\_NO\_DATA** | 12 | Stream contains no data |
| **NPERR\_STREAM\_NOT\_SEEKABLE** | 13 | Seekable stream expected |