# Plugin Development

From Notepad++ Wiki

Jump to: [navigation](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#mw-navigation), [search](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#p-search)

How to develop a plugin or a lexer

## Contents

* [1 **How to install a plugin**](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#How_to_install_a_plugin)
* [2 **How to develop a plugin**](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#How_to_develop_a_plugin)
  + [2.1 Converting plugins](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Converting_plugins)
* [3 Example plugins](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Example_plugins)
* [4 What should a plugin do?](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#What_should_a_plugin_do.3F)
  + [4.1 The Common Plugin Interface](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#The_Common_Plugin_Interface)
    - [4.1.1 The FuncItem structure](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#The_FuncItem_structure)
  + [4.2 Additional lexer library interface](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Additional_lexer_library_interface)
    - [4.2.1 The ILexer interface](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#The_ILexer_interface)
    - [4.2.2 The ILexer object](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#The_ILexer_object)
  + [4.3 The menu initialization and the dll entry point](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#The_menu_initialization_and_the_dll_entry_point)
    - [4.3.1 Menu initialization](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Menu_initialization)
    - [4.3.2 Initialization](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Initialization)
    - [4.3.3 Cleanup](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Cleanup)
* [5 Lexer specific tasks](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Lexer_specific_tasks)
  + [5.1 Meaning and use of the arguments to Lex() and Fold()](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Meaning_and_use_of_the_arguments_to_Lex.28.29_and_Fold.28.29)
  + [5.2 The styler interface](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#The_styler_interface)
  + [5.3 Lexer call initializations](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Lexer_call_initializations)
    - [5.3.1 Unpacking word lists](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Unpacking_word_lists)
    - [5.3.2 Setting properties](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Setting_properties)
  + [5.4 Lexer states](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Lexer_states)
    - [5.4.1 How Merge() works](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#How_Merge.28.29_works)
* [6 Build requirements](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Build_requirements)
  + [6.1 Using the C or C++ language](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Using_the_C_or_C.2B.2B_language)
    - [6.1.1 Compilers](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Compilers)
  + [6.2 Not using the C or C++ language](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Not_using_the_C_or_C.2B.2B_language)
    - [6.2.1 Implementing IsLeadByte(char ch)](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Implementing_IsLeadByte.28char_ch.29)
    - [6.2.2 Retrieving a general property](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Retrieving_a_general_property)
    - [6.2.3 Implementing IndicatorFill(int start, int end, int indicator, int value)](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Implementing_IndicatorFill.28int_start.2C_int_end.2C_int_indicator.2C_int_value.29)
* [7 Debugging](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Debugging)
* [8 References](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#References)

### How to install a plugin

The plugin (in the DLL form) should be placed in the \plugins subfolder of the [Notepad++ Install Folder](http://npp-wiki.tuxfamily.org/index.php?title=Notepad%2B%2B_Install_Folder). Configuration files should go in \Plugins\Config. Documentation files should go to \Plugins\Doc\ directory.

These are generic instructions. If the plugin comes with some different procedure, that procedure should be followed instead.

Once you installed the plugin, you can use (and you may configure) it via the menu "Plugins".

### How to develop a plugin

If you want to do your (very simple) plugin without going through the document, just check [Plugin Development Quick Start Guide](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development_Quick_Start_Guide).

A plugin is a dll which is expected to provide some basic functionality required by Notepad++, in addition of the functionality (enhancements) it provides. External lexers for languages the User Defined Language framework cannot handle are plugins which expose additional interface to Notepad++.

The Windows Message System is used to avoid the overhead. It means the plugins system makes available some Notepad++ handles in order to give more flexibility and possibility to the plugin developers. There're 3 window handles available for the moment: The main Notepad++ handle and 2 Scintilla Window handles.

With these 3 window handles, we can do almost everything, just send the appropriate message to the right handle to get what you want.

Check the newest header file [Notepad\_plus\_msgs.h](http://notepad-plus.svn.sourceforge.net/viewvc/notepad-plus/trunk/PowerEditor/src/MISC/PluginsManager/Notepad_plus_msgs.h?view=markup) that you may need to use the new Notepad++ message. It is mirrored on [Messages And Notifications](http://npp-wiki.tuxfamily.org/index.php?title=Messages_And_Notifications).

The interface (the header file) of plugins is written in C for both C/C++ programmer. However, if you translate it in order to develop in another language, please let it be known so that it can be included for the other language development. There are currently plugins written in Visual Basic, Delphi and Oberon. Any language able to interface with C code will do.

There are some plugin templates available:

* [A C/C++ template in both ANSI/Unicode](http://notepad-plus.sourceforge.net/commun/pluginDemoTemplate/NppPluginTemplate.zip)
* [A Delphi template in ANSI mode](http://sourceforge.net/projects/npp-plugins/files/DelphiPluginTemplate/DelphiPluginTemplate%201.0/DelphiPluginTemplate.zip/download)
* [A Delphi template in Unicode mode](http://sourceforge.net/projects/npp-plugins/files/DelphiPluginTemplate/DelphiPluginTemplate%202.0%20UNICODE/DelphiPluginTemplate2.zip/download)
* [A C# template for ANSI plugins](http://artiche.info/npp/NppDotnetPlugin.zip)
* [A .NET template and demo for Unicode builds](http://www.file-upload.net/download-2120709/NppPlugin.NET.v0.3.7z.html)
* [DFPN++, a library to write plugins in Delphi](http://www.cab.i24.cc/projects/npppluginbase/)

You can check the source code of plugins written in other languages: Web Edit (Oberon), Math Plugin (PureBasic).

#### Converting plugins

A side branch of developing plugins is convertins ANSI ones to Unicode. The following note may help:

All the plugins which are compatible with v5.0.3 (which is an ANSI application) will be compatible with ANSI release, but not with Unicode release. As well, all the Unicode plugins (which come with Unicode Notepad++) are not compatible with Notepad++ ANSI mode.

In order to make your plugins compatible with Unicode Notepad++ (Unicode release will be the main stream on the future versions), you should include the newest [PluginInterface.h](http://notepad-plus.svn.sourceforge.net/viewvc/notepad-plus/trunk/PowerEditor/src/MISC/PluginsManager/PluginInterface.h?view=markup) in your project, then compile your project by setting the symbol UNICODE (with #define UNICODE in your header file or set Character Set option from "Use Multi-Byte Character Set" to "Use Unicode Character Set" in Configuration Properties->General via project properties dialog in your VC studio).

Keep one thing in mind : Send all unicode messages to Notepad++ in unicode mode, but send ansi message to scintilla in both ANSI and UNICODE mode. Actually this is all it takes.

The following snippets summarise the adjustments required in HexEditor by J. Lorenz, an advanced plugin:

using namespace std;

#ifdef UNICODE

#define string wstring

#endif

You must indeed prepare for getting Unicode strings and paths from Notepad++. Don't forget BOMs when applicable.

#ifdef UNICODE

WCHAR wText[65];

ListView\_GetItemText(\_hListCtrl, \_pCurProp->cursorItem, \_pCurProp->cursorSubItem, wText, SUBITEM\_LENGTH);

wText[\_pCurProp->cursorPos] = (TCHAR)wParam;

 ::WideCharToMultiByte(CP\_ACP, 0, wText, -1, text, SUBITEM\_LENGTH, NULL, NULL);

#else

ListView\_GetItemText(\_hListCtrl, \_pCurProp->cursorItem, \_pCurProp->cursorSubItem, text, SUBITEM\_LENGTH);

#endif

Again, the trick is to use the right API variant and send/receive strings in the right format. Don't forget using the TCHAR Windows macro type for individual characters, it will help flatten differences out.

#ifdef UNICODE

static WCHAR wText[129] = \_T("\0");

 ::MultiByteToWideChar(CP\_ACP, 0, text, -1, wText, 129);

lvItem->pszText = wText;

#else

lvItem->pszText = text;

#endif

Yet another conversion.

Remember that Scintilla always stores things in ANSI or UTF-8. Please refer to the Scintilla documentation on UTF-8 encoding messages if you wish to directly access Scintilla's buffers.

Here are some more quotes from XBracket Lite, by D. Vitaliy:

void CXBrackets::OnNppSetInfo(const NppData& nppd)

{

m\_PluginMenu.setNppData(nppd);

isNppWndUnicode = ::IsWindowUnicode(nppd.\_nppHandle) ? true : false;

}

* This one helps talking with Notepad++:

if ( isNppWndUnicode )

{

nppOriginalWndProc = (WNDPROC) SetWindowLongPtrW(

m\_nppMsgr.getNppWnd(), GWLP\_WNDPROC, (LONG\_PTR) nppNewWndProc );

}

else

{

nppOriginalWndProc = (WNDPROC) SetWindowLongPtrA(

m\_nppMsgr.getNppWnd(), GWLP\_WNDPROC, (LONG\_PTR) nppNewWndProc );

### Example plugins

[Plugin Development Quick Start Guide](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development_Quick_Start_Guide) is devoted to them.

Plugin template for Delphi is now available, as well as a few others, from the [| NppPlugins SourceForge project](http://sourceforge.net/projects/npp-plugins). Also check [Plugin Development forum](http://sourceforge.net/forum/forum.php?forum_id=482781%7C) for more plugins, or [Plugin Central](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Central).

You may wish to read [Analysing Plugin Code](http://npp-wiki.tuxfamily.org/index.php?title=Analysing_Plugin_Code) for an example of how and why the code of an advanced, actual plugin is laid out the way it is.

### What should a plugin do?

A plugin is expected to implement some functions. Their names and semantics are explained below.

Further, there are some duties plugins are expected to perform on startup for proper operation. What lexers need to do is significantly more complex, because they have to initialise all sorts of constants that the lexer and folder functions will use.

#### The Common Plugin Interface

bool isUnicode()

A plugin is designed to either work with an ANSI or Unicode build of Notepad++. ANSI plugins must not define this function. Unicode plugins must define it, and it must return true.

void setInfo(NppData)

This routine is called when the plugin is loaded, providing it with information on the current instance of Notepad++ – namely, an array of three handles for:

1. the main Notepad++ window
2. the primary Scintilla control
3. the secondary Scintilla control

Any NPPM\_ or WM\_ message can be sent to the main Notepad++ window. Any WM\_ and SCI\_ message can be sent to either Scintilla control. The Scintilla documentation recommends against the use of EM\_ messages, though a subset is emulated.

const TCHAR\* getName()

Returns name of the plugin, to appear in the Plugin menu.

FuncItem \*getFuncsArray(int \*)

Retrieves a pointer to an array of structures that describe the exposed functions. The expected length of the array is the value pointed by the argument. There must be at least one such routine. Provide one that displays some sort of About dialog box if there is otherwise no need for a menu entry - a typical case for external lexers.

void beNotified(SciNotification \*notif)

This procedure will be called by Notepad++ for a variety of reasons. The complete list of codes is to be found on the [Messages And Notifications](http://npp-wiki.tuxfamily.org/index.php?title=Messages_And_Notifications). It should handle these tasks using information passed in the notification header.

LRESULT messageProc(UINT message, WPARAM wParam, LPARAM lParam)

This is a message processor handling any message Notepad++ has to pass on.

The last two routines, if not needed, may have an empty body, but they *must* be defined.

Each of the above functions must be defined with the following link information, couched in C++ terms: <source lang="c++"> extern "C" \_\_declspec(dllexport) </source> If the plugin is not being developed in C++, consult your compiler or interpreter documentation to find out how to generate a .DLL that conforms with those expectations. E.g. for XDS Oberon that would be something like: PROCEDURE ["C"] isUnicode\*(): BOOLEAN.

##### The FuncItem structure

This structure describes items to add to the plugin's submenu. It is laid out as follows:

char \_itemName[64]

name to display

\*void() \_pfunc

a poiner to the routine to execute, which has no arguments and returns nothing

int \_cmdId

a command index

bool \_init2check

set if the item is checked at startup

Shortcut \*\_pShKey

a pointer to a structure describing the key bound to this item.

The Shortcut structure is as follows:

1. bool \_isCtrl
2. bool \_isAlt
3. bool \_isShift
4. UCHAR \_key

The value for \_key is the desired Windows virtual key code. Note that you cannot use shortcuts of the form: Win[+modifiers]+<Key>.

#### Additional lexer library interface

int getLexerCount()

Returns how many lexers this dll hosts. If the plugin doesn't have any lexer capability, then that function should not be defined at all. This number should be no greater than 30, though there is no bound check.

void getLexerName(int i, TCHAR \*name, int nameLen)

Retrieves the short name for the lexer language of index i in the plugin. Such names should be 12 character long or less. "Lua", C++" or "Verilog" are examples of such names. The name will be used as a foreign key to retrieve settings from langs.xml and stylers.xml. The function must fill the name buffer, whose capacity is nameLen. Attempts to redefine built-in languages are ignored.

void getLexerStatusText(int i, TCHAR \*desc, int descLen)

Returns a string that will be displayed in the status bar when Notepad++ highlights a document using this lexer. "Visual Basic module" could be such a string. Otherwise, arguments have the same meaning as for getLexerName().

LexerFactoryFunction \*GetLexerFactory(int index)

Returns a function pointer that will be called to create the lexer. If your lexer supports several (variants of) languages, use the index parameter to differentiate between them. The function will be called once per valid language index.   
The returned function doesn't take parameters, and returns an ILexer\* representing the requested lexer.

All the above function must be declared with [\_\_stdcall](http://en.wikipedia.org/wiki/X86_calling_conventions#stdcall) calling convention]. Using the C/C++ SCI\_METHOD marker macro adds these specifications to the marked method. If the plugin is not developed in C, then consult your interpreter or compiler documentation in order to generate a dll that conforms with those expectations. E.g. for XDS Oberon that would be something like PROCEDURE ["StdCall"] getLexerCount\*(): LONGINT.

There must exist a <module\_name>.xml file in the plugins\config subfolder of the Notepad++ install folder. This file contains information in the same format as the one found in langs.xml and stylers.xml, with one Language and one LexerType node per language in the dll.

For an example for a complete internal lexer see the [LexCPP.cxx](http://scintilla.cvs.sourceforge.net/viewvc/scintilla/scintilla/src/LexCPP.cxx?view=markup).

##### The ILexer interface

An ILexer is an interface rather than an object. This means it doesn't have a state of its own, just function slots for your lexer to implement:

* If using the C++ language, all you have to do is to create a class inheriting from ILexer and implement all the virtual methods.
* Otherwise, in addition, you'll need to create a low level stub that Scintilla will call.

You can check the lexlib\LexerBase.cxx Scintilla source file for default implementation of most of the functions enumerated below.

int Version()

Supported version of the IDocumet interface. As of Scintilla v2.21, only 0 is a valid version, so return 0.

void Release()

Called on destroying the lexer. Release all additional memory or resource that may have been grabbed.

const char \* SCI\_PropertyNames()

Returns the newline ("\n") separated list of all properties the lexer understands. Return "" if there are none. Before v2.20, this was part of the GetProperties() returned value.

int PropertyType(const char \*name)

Returns the type for the property name supplied. Supported values are SC\_TYPE\_BOOLEAN (0), SC\_TYPE\_INTEGER (1) and SC\_TYPE\_STRING (2). Return 0 if there is no such property.

const char \* DescribeProperty(const char \*name)

Returns a textual description of the property being requested. Since the common language of the Internet is English, it may be a good idea for the returned text to be in English.

int PropertySet(const char \*key, const char \*val)

Assign value to the property. Returns -1 if the key already exists with the same value. Does nothing if key is "".

const char \* DescribeWordListSets()

Returns a newline ("\n") separated list of descriptions for the word lists in lexer.

int WordListSet(int n, const char \*wl)

Sets word list number n to the string wl.

void Lex(unsigned int ext\_index, unsigned int startPos, int lengthDoc, int initStyle,   
   char\*\* words, unsigned int Window\_ID, char\* PropSet)

This function is called by the Scintilla control whenever it needs to colorize some text. The startPos is guaranteed point to the beginning of a document line.

void Fold(unsigned int ext\_index, unsigned int startPos, int lengthDoc, int initStyle,   
   char\*\* words, unsigned int Window\_ID, char\* PropSet)

This function is called at the end of each styled line. It is meant to tell the styler what the level of the current line should be, with a couple flags or'ed in. Defining this routine is optional - folding will be supported only if it is defined.

void\* PrivateCall(int index, void \*data)

Used for coupling the lexer with an application. When Scintilla receives the SCI\_PRIVATECALL message, it passes both parameters to the function and forwards the returned value. The default implementation return (void \*)0.

As with the lexer interface functions, all these must be declared as following the stdcall calling convention.

Most internal lexers still use the pre-v2.20 interface, but lexers loaded from an external dll must comply with the interface above. Their source is still a good source of inspiration for lexing and folding routines.

##### The ILexer object

If you are not using C++ to develop the lexer, you need to manufacture an ILexer object the address is to be returned by the various factory functions. However, this is easy enough:

1. Define an array of pointers containing the addresses of the functions listed in the section above. They must appear in the array in exactly the same order as they do in the interface description.
2. Define a variable of a pointer type, and set it to the address of the array just created. This variable is the ILexer object.

#### The menu initialization and the dll entry point

A dll always has an BOOL APIENTRY DllMain(HANDLE hModule, DWORD reasonForCall, LPVOID lpReserved) entry point so as to manage attaching to and detaching from processes and threads.

##### Menu initialization

Since all plugins must provide a FuncItem[] array of length at least one, this array must be initialised before Notepad++ has loaded the plugin. Since it does so by loading the plugin dll, the only place to do that is the DllMain() entry point.

Typical code will take action only on the ATTACH\_PROCESS reason, and should initialise the array of function descriptions (and optional shortcut) then.

Note that as of 6.6.6 cascading menus are not supported for plugins.

##### Initialization

It is not safe to assume that your dll will be attached only once per instance of Notepad++.exe. Collaborative plugins may need to attach it so as to get information they need.

By definition, an initialization is to be performed only on the first attachment of a given instance of Notepad++. It is only sloppy coding to perform it on each attach if it has no side effect. If it has, then this can cause errors of various kinds.

This is particularly true if the plugin attempts to extend or change the main Notepad++ interface, like adding an extra menu to the main menu bar. Such tasks must have be completed before Notepad++ broadcasts NPPN\_READY to signal the end of the startup process. The first attach event for your dll is the time to do the work.

As an exception, registering icons for the toolbar has to wait for receiving the NPPN\_TBMODIFICATION. Any initialisation related to files being loaded has to wait for tNPPN\_FILEBEFORELOAD or NPPN\_FILELOADED notification.

##### Cleanup

If your plugin needs to do some cleanup, like destroying an icon handle it owns, this should be done on the last process detach for any given instance of Notepad++.exe. The safest way is to keep a reference count of your dll inside its DllMain() entry point. Keeping track of the count will allow to perform initialization only on the first attach and cleanup only on the last detach, the latter being usually more important.

### Lexer specific tasks

Lexers must define states which say how a character is to be interpreted. The word categories and styles are read from langs.xml and stylers.xml, so that it is good practice to define literal symbols that represent those styles.

#### Meaning and use of the arguments to Lex() and Fold()

1. ext\_index is the 0-based index of the lexer being called in the library.
2. startPos is the position at which styling starts.
3. lengthDoc is how long should the lexer proceed, in characters;
4. initStyle is the state of the lexer before the first character of the first line to style;
5. words is the address of an array of pointers to arrays of char, the keywords in the language arranged in categories. The list terminates with NULL;
6. Window\_ID is the handle of the window viewing the document being styled.
7. PropSet is a string holding all Scintilla-wide properties. It is a string like "key1=value1\nkey2=value2\n....\0. As usual, \n is ASCII 0Ah, and \0 is ASCII 0.

#### The styler interface

If you are using C or C++ to develop the lexer, you may take advantage of the Accessor document interface.You create such an object calling the Accessor constructor with an instance of the IDocument class - this instance is private - and a PropSetSimple object, which is public.

|  |  |  |
| --- | --- | --- |
| Accessing the methods of the styler parameter | | |
| **Returns** | **Prototype** | **Description** |
| char | SafeGetCharAt(int position, char chDefault=' '); | Returns character at given position if valid, and default value if not valid. Attempts to fill buffer with valid values prior. |
| bool | IsLeadByte(char ch) | Returns true if character is a lead byte in a DBCS script. |
| bool | Match(int pos, const char \*s); | Returns true if string s can be read from position pos |
| char | StyleAt(int position); | Returns the style byte for that position |
| int | GetLine(int position); | Returns the document line number |
| int | LineStart(int line); | Get the position at which a line starts. |
| int | LevelAt(int line); | Returns the current folding level of the line |
| int | Length(); | Returns document length. |
| void | Flush(); | Reset accessor and invalidate internal buffers |
| int | GetLineState(int line); | Returns the current lexer state |
| int | SetLineState(int line, int state); | Sets the lexer state for this line. |
| void | StartAt(unsigned int start, char chMask=31); | Defines the starting position for styling |
| void | SetFlags(char chFlags\_, char chWhile\_); | Sets internal flags ColourTo(pos, chAttr) uses for optimization:   * if chAttr is not chWhile, chFlags is cleared and chAttr left unchanged * otherwise, chAttr is or'ed with chFlags |
| unsigned int | GetStartSegment(); | Returns starting point of segment. A segment is a range of positions with the same styling. |
| void | StartSegment(unsigned int pos); | Set start of current segment at pos. |
| void | ColourTo(unsigned int pos, int chAttr); | Request colouring a range of characters. |
| void | SetLevel(int line, int level); | Sets folding level for the line. |
| int | IndentAmount(int line, int \*flags, PFNIsCommentLeader pfnIsCommentLeader = 0); | Determines the amount of indentation on a line, as well as a couple flags |
| void | IndicatorFill(int start, int end, int indicator, int value); | Fills a range using an indicator. |
| void | ChangeLexerState(int start, int end) | Signals that a section of the document needs restyling. Scintilla will respond by sending a modification notification to Notepad++. |

Obviously, not all methods are equally useful. The ones that you will always use are ColourTo(), SafeCharacterAt(), LevelAt() and SetLevel().

Access to property values of the lexer is done through the pprops member.

#### Lexer call initializations

1. Unpack the word lists, if you are using them. It is of course more efficient to do this when the lexer is loaded, if they don't change.
2. You are going to send many Scintilla messages to Window\_ID. You can do so by sending the messages to the Scintilla handle, which you know from the PluginInfo structure Notepad++ provided your plugin with on startup. But this has an overhead you can drastically cut. Instead,
   1. Quey SCI\_GETDIRECTPOINTER to get the\_pointer you'll be perusing
   2. Query SCI\_GETDIRECTFUNCTION. You'll get a function pointer, the\_function. Its prototype is as follows (see [scintilla.h](http://scintilla.cvs.sourceforge.net/viewvc/scintilla/scintilla/include/scintilla.h?view=markup)): it returns a long and takes as arguments:
      1. the\_pointer
      2. message\_ID, an integer
      3. wParam, an unsigned integer
      4. lParam, a pointer
   3. Now, to send a message, call (the\_function)(the\_pointer,message\_id,wParam,lParam). Types are obvious.
3. Since you must handle any DBCS in comments or literals, you must obtain the current code page by querying SCI\_GETCODEPAGE.
4. Set the initial styling position by calling SCI\_STARTSTYLING with wParam=start position, lParam=0x1F(=31). This last value corresponds to a compatibility kludge between old and new indicators.
5. In case initStyle is not set properly, check the style value at the end of the previous line. If you find a different value, override the initStyle you were passed.

##### Unpacking word lists

If using the C++ language, you may want to take advantage of the helper functions in the WordList class. To do this, count how many entries the words array has, define an array of WordLists of the same size, and initialise each using the corresponding char\* pointer.

##### Setting properties

If your lexer needs to set properties on the Scintilla component, you must be aware that the component may change because of switching views, or because of other plugins creating extra handles. You must be prepared to find a component without the properties you expect, in which case you should set them according to their current state out of consistency with other views.

#### Lexer states

Scintilla has provided a single machine sized integer for each document line, accessed using the SCI\_GETLINESTATE and SCI\_SETLINESTATE messages. This integer is for the lexer private use only. While certainly useful, this is not enough to capture more complex states, like a hierarchy of folding blocks of code. Various workarounds exist, but they are not always satisfactory:

* You can reckon a state by backtracking until you rebuild the information. This can be inefficient, and fixed size backtracking windows introduce glitches of their own.
* Use the integer to store a pointer. This won't work on machines where pointers and integers don't have the same size - most 64 bit machines -. Further, Scintilla doesn't notify the lexer of deleted lines, so memory leaks are guaranteed.
* Use lnined lists of states, using the line state as index into the list: while this is workable in theory, it still incurs the overhead of checking whether the linkage was modified so as to remove obsolete states.

Since v2.25, Scintilla provides a new structure, called SparseState, that you can use to detect whether some complex state has changed. Since it is a template (the C++ word for generic parameters -, using it in lexers not written in C++ could prove tricky, and won't be dealt with here.

A SparseState<T> is a structure holding a state of type T. The use model is as folows:

1. Create a fresh structure on starting an invocation of Colourise()
2. Jot down state related information in this structure as you go
3. Merge it with the previous state to determine whether complex state changed.

The SparseState<T> structure has the following public features:

|  |  |
| --- | --- |
| The generic SparseState<T> structure | |
| **Member or method** | **Meaning or usage** |
| SparseState(position) | Create a new object with an initial position. A position is an integher, the expected values of which are not negative. If argument is omitted, -1 is assumed. |
| void Set(position , value) | Adds a new state information (value) concerning a position (position). |
| bool Delete(position) | Remove the state information for the provided position and beyond. Returns true if there as anything to delete, else false. |
| size\_t size() | Returns how many states the structure holds. |
| T ValueAt(position) | Returns the state prevailing at position. If there is a syaye declared below the position, this state is returned. Otherwise, whatever T() produces is returned. |
| bool Merge(const SparseState<T>& other, int ignoreAfter) | Reckons whether adding the new states causes significant change, and returns true if and only if so. |

Positions need not be Scintilla text positions in a document, though this is the obvious thing to do. It is up to you to decide what positions mean.

The type T can be anything, as long as it has a no-argument constructor. Disposing of old states is done when you dispose of SparseState's, so it is up to you to manage them. The basic model is to have one such structure for each document, merging local information as it comes. Any likeness with distributed >SCMs using local and remote branches is probably intended.

##### How Merge() works

First of all, whatever lies past the ignoreAfter position in the target structure is Deleted(). This mimicks the max line state property which Scintilla uses to manage integer line states.

Assuming the two structures aren't obviously equal, the following processing is performed:

* If the first state change in the other structure is not beyond the final state of the target structure, the latter is Delete()d from that position, and true will be returned. Indeed these deleted states have been probably invalidated by changes recorded in the other structure.
* After this invalidation step, the two structures are concatenated. true is returned if this changed the target structure; otherwise and if no invalidation, false.

Note: this documentation comes from reading the source code. No material is posted on Scintilla's site yet.

### Build requirements

#### Using the C or C++ language

You will have to include some Scintilla code in order to call methods on theDocumentAccessor object the lexer and folder functions are passed.

The following list of include files seems to be more than enough. Depending on your own requirements, some may be removed:

* #include <stdlib.h>
* #include <string.h>
* #include <ctype.h>
* #include <stdio.h>
* #include <stdarg.h>
* #include <assert.h>
* #include <windows.h>
* #include "Platform.h"
* #include "ILexer.h"
* #include "LexAccessor.h"
* #include "Accessor.h"
* #include "StyleContext.h"
* #include "WordList.h"
* #include "Scintilla.h"
* #include "SciLexer.h"
* #include "CharClassify.h"
* #include "LexerModule.h"
* #include "PropSetSimple.h"

Scintilla and Notepad++ are compiled using the Visual Studio C++ compiler (version 2008 and 2010 are supported). You can use the [Plugin Interface Library](http://cid-71a486ea08eb407c.skydrive.live.com/browse.aspx/.Public/NppPluginIface) by [aathell](http://sourceforge.net/users/aathell/) in order to automate some initialisation chores and more. Since this library has not been updated after Scintilla changed the lexer interface, it is better to borrow code from it, but including it single-handedly will cause plugin crashes.

##### Compilers

Here is a happy user how-to for using MinGW instead:

1. First I had to convert the Visual Studio .vcproj project file to the old Visual Studio .dsp format using the [i]prjconverter.exe[/i] linked in <http://forum.icd.no/viewtopic.php?t=64> .
2. I followed the rest of the instructions in that page to generate a makefile from the .dsp project.
3. Once the makefile was generated, it was ready to build the plugin with minor tweaking. By default it will generate an ANSI plugin no longer compatible with current versions on Notepad++, so you have to change the CFG target.

#### Not using the C or C++ language

The following table loosely follows the interface listing for a document accessor, and shows alternate methods to obtain the same result. The table is made of alternating rows: even numbered ones hold comments. The leftmost part of odd numbered rows describes a task. The right part displays the message to send and its parameters, or some other indication when sending a single message wouldn't do.

|  |  |
| --- | --- |
| Emulating the methods of a document accessor | |
| Getting a character at some position | |  |  | | --- | --- | | SCI\_GETCHARAT | | | wParam: | Document position | | lParam: | 0 | |
| Returns the character or 0. If the text can hold NULs, for instance because it has comments using a DBCS script, this is not safe enough, because 0 may be valid | |
| Get some range of text | |  |  | | --- | --- | | SCI\_GETTEXTRANGE | | | wParam: | 0 | | lParam: | pointer to: start position, end position, address of buffer. The end position can be -1 to go all the way to end of document. | |
| Returns the number of characters copied. The buffer should also accommodate an extra 0 terminator (not counted in return value). It is a good policy to fill the buffer at a position sligtly before the actual one, so as to perform any needed backtracking. | |
| Decide whether a character is a DBCS lead byte | |  |  | | --- | --- | | [#Implementing IsLeadByte(char ch)](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Implementing_IsLeadByte.28char_ch.29) | | | wParam: | N/A | | lParam: | N/A | |
| You must inquire the current code page using the SCI\_GETCODEPAGE message. | |
| Decide whther a string is a substring of buffer contents | |  |  | | --- | --- | | Not available | | | wParam: | N/A | | lParam: | N/A | |
| Use the string methods of your language. | |
| Get the style at some position | |  |  | | --- | --- | | SCI\_GETSTYLEAT | | | wParam: | position | | lParam: | 0 | |
| Returns the style byte for that position, or 0 if position is invalid. | |
| Retrieve line number for a given position | |  |  | | --- | --- | | SCI\_LINEFROMPOSITION | | | wParam: | position | | lParam: | 0 | |
| Returns the document line number. | |
| Retrieve start position for a line | |  |  | | --- | --- | | SCI\_POSITIONFROMLINE | | | wParam: | line number | | lParam: | 0 | |
| Returns <p>   * the document position at which line starts, if valid * the doucment position the line holding current selection starts, if line<0 * -1 if line>document line count * end of document position if line = line count | |
| Retrieve end position for a line | |  |  | | --- | --- | | SCI\_LINEENDPOSITION | | | wParam: | line number | | lParam: | 0 | |
| Returns position of last non terminating character on the line. The line number must be valid. | |
| Retrieve the length of a line | |  |  | | --- | --- | | SCI\_LINELENGTH | | | wParam: | line number | | lParam: | 0 | |
| Returns the line length, including any terminating characters. Returns 0 if line is not valid. | |
| Retrieve the folding level for a line | |  |  | | --- | --- | | SCI\_GETFOLDLEVEL | | | wParam: | line number | | lParam: | 0 | |
| Returns the current folding level of the line | |
| Retrieve document length | |  |  | | --- | --- | | SCI\_GETLENGTH | | | wParam: | 0 | | lParam: | 0 | |
| Returns document length. | |
| etrieve the state for the current line | |  |  | | --- | --- | | SCI\_GETLINESTATE | | | wParam: | line | | lParam: | 0 | |
| Returns the current lexer state. If the line doesn't have a state, this is undefined. | |
| Determine the largest line number for which the state is known | |  |  | | --- | --- | | SCI\_GETMAXLINESTATE | | | wParam: | 0 | | lParam: | 0 | |
| Returns the largest line number for which a state is known. | |
| Sets the state for the current line | |  |  | | --- | --- | | SCI\_SETLINESTATE | | | wParam: | line | | lParam: | new state | |
| States are integer values your lexer is the sole user of. They hold persistent information which mught be usedin a multiple pass parsing, for instance. | |
| Ask for the property list | |  |  | | --- | --- | | SCI\_PROPERTYNAMES | | | wParam: | N/A | | lParam: | [out] char\* names | |
| Fills buffer, if non-zero, with a \n-separated list of supported property names. Returns needed size/copied size. | |
| Read a property value as string | |  |  | | --- | --- | | SCI\_GETPROPERTY | | | wParam: | [in] string key | | lParam: | [out] char \*value | |
| Pass 0 as lParam to retrieve the length of needed buffr. Returns length copied to \*value. | |
| Ask for a property with integer value | |  |  | | --- | --- | | SCI\_GETPROPERTYINT | | | wParam: | pointer to property name | | lParam: | Default value to return if property invalid | |
| Returns the value of the property if found, or the supplied default value else. | |
| Set the style for a range of characters | |  |  | | --- | --- | | SCI\_SETSTYLING | | | wParam: | number of characters to style | | lParam: | style value to apply | |
| Requires a styling position to have been set prior. Styles are defined in the xml companion file. | |
| Set fold level for a line | |  |  | | --- | --- | | SCI\_SETFOLDLEVEL | | | wParam: | line number | | lParam: | fold level | |
| Please refer to the Scintilla docs: the level is made of an index - top level is at 1024 - and or'ed in flags:   * SC\_FOLDLEVELWHITEFLAG: set this for blank lines, so that they can be folded in a smarter way * SC\_FOLDLEVELHEADERFLAG: there is a folder point on this line | |
| Retrieve indentation of a line | |  |  | | --- | --- | | SCI\_GETINDENTATION | | | wParam: | line number | | lParam: | 0 | |
| Returns the number of spaces a line is indented. You must supply any code that would check for indentation consistency or such. Use SCI\_GETINDENT to get the indent step width. If this is 0, the actual width of the step is given by querying SCI\_GETTABWIDTH. | |
| Use indicators on a range of characters | |  |  | | --- | --- | | [#Implementing IndicatorFill(int start, int end, int indicator, int value)](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development#Implementing_IndicatorFill.28int_start.2C_int_end.2C_int_indicator.2C_int_value.29) | | | wParam: | N/A | | lParam: | 0 | |
| Fills a range using an indicator. | |
| Report restyling needed | |  |  | | --- | --- | | SCI\_CHANGELEXERSTATE | | | wParam: | int start | | lParam: | int end | |
| Notepad++ will be sent a text modified notification with these parameters. The notification for event SC\_MOD\_LEXERSTATE needs oto be enabled prior. | |

Note that it is up to your code to perform any segment management if you need it.

##### Implementing IsLeadByte(char ch)

If the current codepage is 0, return 0, else forward the call to the Win32 API IsDBCSLeadingByteEx(ch, codepage).

##### Retrieving a general property

This usually involves two calls to the SCI\_GETPROPERTY message, with wParam the address of a buffer holding the 0 terminated property name.

Since it is safer not to assume anything on the property value length, if only to avoid a buffer overrun exploit, you need to get the length, allocate a buffer of that length and fill it with the desired text. This is done as follows:

1. Call with lParam=0 to get the length of the answer.
2. Allocate a buffer of that length+2 (beware DBCS values of properties)
3. Call again with the buffer address as lParam. The returned string length does not include the 0 terminator.

There is a variant SCI\_GETPROPERTYEXPANDED message. If a property value has a substring like "$(<prop-name>)", then that string is recursively replaced by the value of <prop-name>.

##### Implementing IndicatorFill(int start, int end, int indicator, int value)

You may wish to read the Scintilla documentation on indicators to decide whether you are going to use them.

This is a three stage proess:

1. Call message SCI\_SETINDICATORCURRENT with wParam=indicator, lParam=0 to set the indicator you are going to use
2. Call SCI\_INDICATORVALUE with wParam=value, lParam=0 to set the value for that indicator, i.e. its appearance.
3. Call SCI\_INDICATORFILLRANGE with wParam=start position, lParam=end position to effectively perform the task.

### Debugging

This is a proven fact: everyone else's plugins all have bugs.

Since a plugin interacts with Notepad+ to monitor both Notepad++ code and the plugin code - for simplicity, let's assume interaction between plugins isn't an issue. So, the solution is straightforward:

* create a solution (in Visual Studio parlance) that includes both the Notepad++ project and all the plugin projects.
* All projects must be in debug mode
* Compile the solution
* Now that all the code is under a single umbrella, debug just like any application.

If the plugin is not written in C++, it will have to play with two debuggers, one for each part. The plugin dll must still be included somehow on the C++ side. Who knows, it may even work.

### References

* [Messages And Notifications](http://npp-wiki.tuxfamily.org/index.php?title=Messages_And_Notifications)
* [Notepad\_plus\_msgs.h](http://notepad-plus.svn.sourceforge.net/viewvc/notepad-plus/trunk/PowerEditor/src/MISC/PluginsManager/Notepad_plus_msgs.h?view=markup)
* [PluginsManager.cpp](http://notepad-plus.svn.sourceforge.net/viewvc/notepad-plus/trunk/PowerEditor/src/MISC/PluginsManager/PluginsManager.cpp?view=markup)
* <http://scintilla.sourceforge.net/ScintillaDoc.html>
* <http://scintilla.sourceforge.net/Lexer.txt>
* <http://sphere.sourceforge.net/flik/docs/scintilla-container_lexer.html>
* <http://sphere.sourceforge.net/flik/docs/scintilla-folding.html>

Retrieved from "<http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development&oldid=6482>"

[Categories](http://npp-wiki.tuxfamily.org/index.php?title=Special:Categories):

* [Short Title(All)](http://npp-wiki.tuxfamily.org/index.php?title=Category:Short_Title(All))
* [NP++ Dev](http://npp-wiki.tuxfamily.org/index.php?title=Category:NP%2B%2B_Dev)
* [Programming](http://npp-wiki.tuxfamily.org/index.php?title=Category:Programming)
* [Plugins](http://npp-wiki.tuxfamily.org/index.php?title=Category:Plugins)
* [Lexer](http://npp-wiki.tuxfamily.org/index.php?title=Category:Lexer)
* [Usage(Programming)](http://npp-wiki.tuxfamily.org/index.php?title=Category:Usage(Programming))
* [Usage(NP++ Project)](http://npp-wiki.tuxfamily.org/index.php?title=Category:Usage(NP%2B%2B_Project))

## Navigation menu

### Personal tools

* [Create account](http://npp-wiki.tuxfamily.org/index.php?title=Special:UserLogin&returnto=Plugin+Development&type=signup)
* [Log in](http://npp-wiki.tuxfamily.org/index.php?title=Special:UserLogin&returnto=Plugin+Development)

### Namespaces

* [Page](file:///I:\Projects\VC\ContentsViewerPlugin\Release\Notepad++\Plugin%20Development%20-%20Notepad++_files\Plugin%20Development%20-%20Notepad++.htm)
* [Discussion](http://npp-wiki.tuxfamily.org/index.php?title=Talk:Plugin_Development&action=edit&redlink=1)

### Variants

### Views

* [Read](file:///I:\Projects\VC\ContentsViewerPlugin\Release\Notepad++\Plugin%20Development%20-%20Notepad++_files\Plugin%20Development%20-%20Notepad++.htm)
* [View source](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development&action=edit)
* [View history](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development&action=history)

### Actions

### Search

窗体顶端



窗体底端

### NpWiki++

* [Main Page](http://npp-wiki.tuxfamily.org/index.php?title=Main_Page)
* [Help writing articles](http://npp-wiki.tuxfamily.org/index.php?title=Writing_Articles)
* [Tools and Helpers](http://npp-wiki.tuxfamily.org/index.php?title=Category:NpWiki%2B%2B)
* [Recent changes](http://npp-wiki.tuxfamily.org/index.php?title=Special:RecentChanges)

### NP++ ARTICLES

* [by Title](http://npp-wiki.tuxfamily.org/index.php?title=Category:Short_Title(All))
* [by Topic](http://npp-wiki.tuxfamily.org/index.php?title=Category:Keywords)
* [by Usage](http://npp-wiki.tuxfamily.org/index.php?title=Notepad%2B%2B_Articles_Index_-_Class_Usage)
* [by Area](http://npp-wiki.tuxfamily.org/index.php?title=Area_Index_For_Notepad%2B%2B_Articles)

### NP++ RESOURCES

* [Guides](http://npp-wiki.tuxfamily.org/index.php?title=NP%2B%2B_Guides)
* [Plugins](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Resources)
* [Project Support](http://npp-wiki.tuxfamily.org/index.php?title=Project_Resources)
* [User community](http://npp-wiki.tuxfamily.org/index.php?title=Category:User_Community)

### NP++ LINKS

* [Gateway (Home)](http://notepad-plus-plus.org/)
* [News](http://notepad-plus-plus.org/news/)
* [Latest releases](http://sourceforge.net/apps/mediawiki/notepad-plus/index.php?title=Releases#Version_5_series)
* [Forums](http://sourceforge.net/projects/notepad-plus/forums)
* [Downloads](http://notepad-plus-plus.org/download)
* [IdeaTorrent](http://sourceforge.net/apps/ideatorrent/notepad-plus/ideatorrent/)

### INVISIBLE

### Tools

* [What links here](http://npp-wiki.tuxfamily.org/index.php?title=Special:WhatLinksHere/Plugin_Development)
* [Related changes](http://npp-wiki.tuxfamily.org/index.php?title=Special:RecentChangesLinked/Plugin_Development)
* [Special pages](http://npp-wiki.tuxfamily.org/index.php?title=Special:SpecialPages)
* [Printable version](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development&printable=yes)
* [Permanent link](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development&oldid=6482)
* [Page information](http://npp-wiki.tuxfamily.org/index.php?title=Plugin_Development&action=info)
* This page was last modified on 6 May 2014, at 14:31.
* This page has been accessed 316,390 times.
* Content is available under [GNU Free Documentation License 1.3 or later](http://www.gnu.org/copyleft/fdl.html) unless otherwise noted.
* [Privacy policy](http://npp-wiki.tuxfamily.org/index.php?title=NppWiki:Privacy_policy)
* [About Notepad++ Wiki](http://npp-wiki.tuxfamily.org/index.php?title=NppWiki:About)
* [Disclaimers](http://npp-wiki.tuxfamily.org/index.php?title=NppWiki:General_disclaimer)