Messenger Plus! Plug-in Authors' Guide

Introduction

Welcome to the 'Messenger Plus! Plug-in Authors' Guide'!

This guide is designed to supplement the commented API code provided by Patchou. While this is designed to be easy to understand, it does depend on the users of this guide to be familiar with the concepts of programming in general, and their chosen language in particular. It also assumes the user is familiar with Messenger and Plus, and understands the ReadMe file that comes with the API.

The Messenger Plus! API

There exist two APIs in the download from the Messenger Plus website, one for C++ authors, and one for Visual Basic coders. The VB API, due to the greater simplicity of Visual Basic, is somewhat different to the C++ API, and so will be considered separately.

Messenger Plus! C++ API

Functions

Any API has functions that can be called, and the Plus API is no exception. There are 7 functions that can be used:

- Initialize
- Uninitialize
- PublishInfo
- ParseCommand
- ParseTag
- ReceiveNotify
- DisplayToast

At the absolute minimum, any plug-in has to implement Initialize, and will probably implement a Parse function too. DisplayToast is already implemented in the API.

Windows NT4/2000/XP only: For Unicode compatible plugins, all functions but Initialize and Uninitialize can be suffixed by a W to implement Unicode compatibility. However, this is not required, as Plus will handle the necessary character conversions and use the standard functions instead. Initialize and Uninitialize do not use Unicode, so they cannot have a 'W' appended.

Each of the functions here will be presented from the point of view of a C++ programmer. VB programmers don't have access to most of C++'s exotic datatypes, so it has two extra functions which will be dealt with separately.

Initialize

Called when the user signs into Messenger. Use to set up the plug-in, and any needed resources.

Arguments:

Name	*Direction*	*Description*
DWORD nVersion	In	Don't let the DWORD put you off: this is merely an integer representing the API version.
const char* sUserEmail	In	The e-mail address of the user's Passport.
IDispatch* iMessengerObj	In	This exposes (part of) Messenger's own API. This allows you to perform operations on Messenger itself. Most plugins will not need to be concerned with this.

Return value: A Boolean indicating success. If the return value is FALSE, the plug-in cannot be used.

Uninitialize

Called when the user signs out. Use to free any resources you may be using.

Arguments: none.

Return value: none.

PublishInfo

Allows Plus to add an entry for your plug-in on the Plus menu in a conversation window. Called only if Initialize returns TRUE.

Arguments:

Name	*Direction*	*Description*
char* sPluginName	Out	A string containing the name of the plug-in
PLUGIN_PUBLISH_LIST* aCommands	Out	A list of commands the plug-in recognises
PLUGIN_PUBLISH_LIST* aTags	Out	A list of tags the plug-in recognises

Return value: A Boolean indicating whether the plug-in is usable or not. This is more a formality, as not many plugins will have trouble providing this information.

ParseCommand

Called by Plus every time a command is issued (of the form /x*).

Arguments:

Name	*Direction*	*Description*
const char* sCommand	In	The command used (including the /x), converted to lower case
const char* sCommandArgs	In	All the text that followed the command
PLUGIN_PARAM* pParam	In	Extra parameters that may be useful
char* sResult	Out	A 4096-character buffer to receive output

Return value: A Boolean indicating if your plug-in understood the command successfully.

ParseTag

Called by Plus every time there is a tag of the form (!X*) or (!x*).

Arguments:

Name	*Direction*	*Description*
const char* sTag	In	The full tag used, case preserved
PLUGIN_PARAM* pParam	In	Extra parameters that may be useful
char* sResult	Out	A 2048-character buffer to receive output

Return value: A Boolean indicating if your plug-in understood the tag successfully.

ReceiveNotify

Called by Plus if the received message starts with a special control character.

Arguments:

Name	*Direction*	*Description*
const char* sNotifyCode	In	A five-letter code representing the command used
const char* sText	In	All the text that followed the command
PLUGIN_PARAM* pParam	In	Extra parameters that may be useful
char* stextToSend	Out	A 4096-character buffer for an optional response

Return value: A Boolean indicating successful interpretation of the notification.

DisplayToast

This function is only available from Plus v2.50 or later.

This function can be used to display a toast pop-up, similar to the ones used by Messenger itself. Unlike all the other functions, this is already implemented. Therefore, it can be used just like any C++ function.

Arguments:

Name	*Direction*	*Description*
const char* sMessage	In	The message to display
const char* sTitle	In	The title of the toast pop-up
const char* sProgram	In	If not NULL, clicking the toast will cause this to be treated as a URL or a program file to be run
bool bPlaySound	In	If TRUE, Messenger will play the default sound in accompaniment

Return value: none.

Unicode

As mentioned before, five of the functions have Unicode versions, which have the same names but with a 'W' appended. These will only be used on Windows 2000/XP systems, and are not compulsory. If not present, Plus will call the standard verisons instead.

If you wish to use the Unicode versions though, there are a few datatypes that will need changing:

ANSI version	*Unicode version*
char	WCHAR
PLUGIN_PUBLISH_LIST	PLUGIN_PUBLISH_LISTW
PLUGIN_PARAM	PLUGIN_PARAMW

Otherwise, everything is the same as before.

Structures

There are two structures that plugins can use (the third is internal to DisplayToast): PLUGIN_PUBLISH_LIST and PLUGIN_PARAM.

PLUGIN_PUBLISH_LIST

Members:

Name	*Description*
int nCount	The number of individual data items
char sName[50][128]	A short description of the command or tag
char sValue[50][128]	The actual command or tag (see API for more details)
char sHelp[50][128]	Optional: If your commands use arguments, this can be used to hold argument placeholders

PLUGIN_PARAM

Members:

Name	*Description*
IDispatch* iConversationWnd	A handle to the conversation window from which the function was called
char sContactName[512]	The display name of the user whose action triggered the call

Unicode

The same transformations apply.

Constants

There are several constants that can be used to add colour and formatting to text, as well as for forming special messages that trigger ReceiveNotify or imitate the function of the '/me' command.

Name	*Position*	*Meaning*
sCCNotify	Beginning	Use to trigger the ReceiveNotify function
sCCColor	Anywhere	Colours the text following (works like the (!FCx) tag)
sCCBold	Anywhere	Bolds the text following
sCCItalic	Anywhere	Italicises the text following
sCCUnderline	Anywhere	Underlines the text following
sCCStrikeout	Anywhere	Strikes out the text following
sCCReset	Anywhere	Used to reset the formatting back to the user's choice for all following text
sCCNoIcon	Beginning	Prevents emoticons from being displayed
sCCMeStart	Beginning	Used together to imitate the function of the '/me' command
sCCMeEnd	End	

Messenger Plus! Visual Basic API

The Visual Basic API differs from the C++ API in a few respects:

1. There are less complicated datatypes
2. There are no structures
3. PublishInfo is replaced by three new functions
   - GetPublishInfo
   - GetPublishCommandInfo
   - GetPublishTagInfo
4. All the constants apart from sCCMeStart and sCCMeEnd are integer values, and start with nCC instead of sCC

Functions

It is worth covering these again, as there are some significant changes. It is also worth pointing out that there are no Unicode versions of the functions in the VB API, as VB has no built-in support for Unicode.

Initialize

Called when the user signs into Messenger. Use to set up the plug-in, and any needed resources.

Arguments:

Name	*Direction*	*Description*
Long nVersion	In	An integer representing the API version.
String sUserEmail	In	The e-mail address of the user's Passport.
Object oMessenger	In	This exposes (part of) Messenger's own API. This allows you to perform operations on Messenger itself. Most plugins will not need to be concerned with this. The type is 'Messenger'

Return value: A Boolean indicating success. If the return value is FALSE, the plug-in cannot be used.

Uninitialize

Called when the user signs out. Use to free any resources you may be using.

Arguments: none.

Return value: none.

GetPublishInfo

Allows Plus to add an entry for your plug-in on the Plus menu in a conversation window. Called only if Initialize returns TRUE.

Arguments:

Name	*Direction*	*Description*
String sPluginName	Out	Name of the plugin
Integer nCommandCount	Out	Number of commands
Integer nTagCount	Out	Number of tags

Return value: A Boolean indicating whether the plug-in is usable or not. This is more a formality, as not many plugins will have trouble providing this information.

GetPublishCommandInfo

Allows Plus to add an entry for each command on the Plus menu in a conversation window. Called only if Initialize returns TRUE, and nCommandCount > 0.

Arguments:

Name	*Direction*	*Description*
Integer nCommandIdx	In	A count of the number of times the function has been called
String sName	Out	Name of command
String sValue	Out	The command itself
String sHelp	Out	Argument placeholders

Return value: none.

GetPublishTagInfo

Allows Plus to add an entry for each tag on the Plus menu in a conversation window. Called only if Initialize returns TRUE, and nTagCount > 0.

Arguments:

Name	*Direction*	*Description*
Integer nTagIdx	In	A count of the number of times the function has been called
String sName	Out	Name of tag
String sValue	Out	The tag itself

Return value: none.

ParseCommand

Called by Plus every time a command is issued (of the form /x*).

Arguments:

Name	*Direction*	*Description*
String sCommand	In	The command used (including the /x), converted to lower case
String sCommandArgs	In	All the text that followed the command
Object oConversationWnd	In	Handle to the conversation window. Type is 'MessengerConversationWnd'
String sResult	Out	A 4096-character buffer to receive output

Return value: A Boolean indicating if your plug-in understood the command successfully.

ParseTag

Called by Plus every time there is a tag of the form (!X*) or (!x*).

Arguments:

Name	*Direction*	*Description*
String sTag	In	The full tag used, case preserved
Object oConversationWnd	In	Handle to the conversation window. Type is 'MessengerConversationWnd'
String sResult	Out	A 2048-character buffer to receive output

Return value: A Boolean indicating if your plug-in understood the tag successfully.

ReceiveNotify

Called by Plus if the received message starts with a special control character.

Arguments:

Name	*Direction*	*Description*
String sNotifyCode	In	A five-letter code representing the command used
String sText	In	All the text that followed the command
String sContactName	In	The name of the contact who sent the text
Object oConversationWnd	In	Handle to the conversation window. Type is 'MessengerConversationWnd'
String stextToSend	Out	A 4096-character buffer for an optional response

Return value: A Boolean indicating successful processing of the notification.

DisplayToast

This function is only available from Plus v2.50 or later.

This function can be used to display a toast pop-up, similar to the ones used by Messenger itself. Unlike all the other functions, this is already implemented. Therefore, it can be used just like any C++ function.

Arguments:

Name	*Direction*	*Description*
String sMessage	In	The message to display
String sTitle	In	The title of the toast pop-up
String sProgram	In	If not NULL, clicking the toast will cause this to be treated as a URL or a program file to be run
Boolean bPlaySound	In	If TRUE, Messenger will play the default sound in accompaniment

Return value: Strangely enough, this returns TRUE. This is due to the function mechanism in VB.

The minimal plug-in

The API files come with a sample plug-in written in C++, and one written in Visual Basic. They both follow the same structure, so I will cover the two versions simultaneously.

Initialisation

C++ only: Be sure to include a DllMain function! The one in the MPPlugin sample should be enough for most plugins.

The absolute minimum a plug-in must do in the Initialize function is return TRUE, which is exactly what MPPlugin does. This prompts Plus to call the other functions that are used to set up the plug-in.

Information gathering

While the same information is provided by both plugins, they do it in different ways, due to the differences in the APIs.

C++

The name of the plug-in is entered, and then the commands' and tags' information is entered as well. The code is fairly simple, being mostly calls to strcpy() with a few regular assignments as well.

Visual Basic

Things are only marginally more complicated here, as there are three functions. First, GetPublishInfo is called, which tells Plus the name of the plug-in, as well as how many commands and tags the plug-in provides. Then, GetPublishCommandInfo is called the number of times specified in the variable nCommandCount in GetPublishInfo, and similarly with GetPublishTagInfo.

This particular sample uses If... Then... Else blocks to fill in the data, but any method can be used.

Using the plug-in

OK, now the plug-in is set up. If all the above functions returned TRUE, we can now use the Plus menu to select commands and/or tags provided, or we can type them in.

Commands

For example, if a command is entered, ParseCommand is called. This compares the string sCommand to a series of presets, and then performs an action based on the command. If it is the first command, the plug-in displays a MessageBox. The other commands do something more complicated, so let's look at them separately.

The second command is a good exapmle of using the NotifyCode in a message. The code itself must be five letters long, and prepended with the (s/n)CCNotify special character. The text that goes with it is appended. Note that there are no spaces inserted: the way Plus handles notify codes means that spaces aren't necessary like they are with commands.

The third command demonstrates a feature added in Plus version 2.21: the ability to send multiple messages in one go. Each message is separated by a # character. If you want to override the default timing of 0.5s between messages, use #<delay>#<msg>.

The fourth command demonstrates the toast pop-up feature. This particular call specifies a URL: clicking the resulting toast will take you to http://www.msgplus.net/, a very cool place indeed, but you knew that already, didn't you?

Tags

Tags are processed almost identically to commands, except that only the formatting control characters can be used. The example provided picks one of three replacements at random: not very imaginative, but it is only a sample. However, it does provide a good example of random number generation, which can be very useful...

Notes

Both the Parse functions return TRUE if the understand the command/tag, and FALSE if they didn't. If they didn't do this, it could stop other plugins from working, or prevent itself from working. Always make sure these functions return the correct value.

ReceiveNotify

The Parse functions are used by the sender, so how can the receiver benefit? You guessed it: ReceiveNotify. This function is core to many plugins, including optimism_'s Audiator plug-in, and RaceProUK's sound plugins.

Whenever Plus receives a message that is preceeded by the (s/n)CCNotify code, that character is stripped off, and ReceiveNotify called. For the sample plug-in, that means that it responds to the 'ping' command. It does this by filling the sTextToSend buffer (for an automated reply). Of course, this is not the only thing that it can do...

Uninitialisation

When you just cannot go any longer, you sign off. When you do, all plugins must uninitialise. However, this does not mean they have to do anything: MPPlugin merely defines an empty function. If it used any resources, such as the MCI API, then Uninitialize() is the place to free these resources.

Conclusion

Writing a simple but useful Plus plug-in is quite simple, and with a little bit of thought, planning, and reading up, some quite interesting tasks can be performed. Changing text and playing sounds is just the tip of the iceberg. As inspiration to get started, here are a few tasks plugins can perform:

- Play sounds
- Print the currently playing song in Windows Media Player
- Change the user's screen name
- Send files
- Launch games
- Search Google
- Translate text using Google
- Gather information about your PC
- L33tify, colourise, reverse, obfuscate, etc...
- Just too many things to list!

Code samples

Sound playing plug-in

As a more complete example of a plug-in, this guide has an accompanying package which includes various source code files that demonstrate most if not all the functions in the API. The first

example is lifted straight from a real plug-in, RaceProUK's AllYourBase Sounds Plug-in. This plug-in implements all bar the Tag functions, as it only uses commands.

AllYourBase uses the following resources: Windows Media Player, Win32 API.

What the plug-in does

There are two commands with this plug-in: one shows a message box that displays version information, and the second plays a short PCM WAV file, which is CATS saying 'All Your Base Are Belong To Us'. When the author wrote this plug-in, it was as a testbed for ideas, and the code is easily extendible to more sounds (5, 8, 29, 42...).

Overview of the code

The plug-in is coded using Visual Basic 6 Professional Edition SP5. The filename is AYB.cls.

The best place to start is at the beginning. It is better to read through the code that way, as all the declarations are usually at the beginning. The file is divided up (by VB) into one section per function, and the (Declarations) section at the top of it all.

Global declarations

The (Declarations) section is where all the resources the plug-in uses are declared. The first few delarations are merely variables of specific types: it's always better to have a strongly-typed language: it can cause more headaches, but the final code is more reliable. There may seem to be redundancy in the variable declarations (there are two variables that hold the plug-in path), but there is a very good reason for this.

After the variables come the interesting declarations. The first is a declaration of a reference to a WindowsMediaPlayer. This is what allows the plug-in to play sound. The rest of the declarations are to access the Registry via the Win32 API. For more information on this, visit the MSDN website at http://msdn.microsoft.com/.

Initialisation and Uninitialisation

Now, with all the global declarations sorted out, the real code begins. First, the Initialize function. Take your time here: this is not simple code, as there are some formalities with Registry access. Note all the extra variables: they are all used.

The first line that does something interesting is the call to RegOpenKeyEx. This opens the Registry key specified, in this case HKEY_LOCAL_MACHINE\SOFTWARE\Patchou\MsgPlus2. The KEY_READ defined in the (Declarations) section is used here to specify read-only access. Now, not all Registry operations are successful, so if the returnVal is not a 0 (for success), the plug-in should abort and tell Plus it cannot be used. If the opening was successful (and should be if you have Plus 2.20 or later installed), the plug-in then attempts to read a value, in this case 'PluginDir'.

Do you see why there are two strings for the plug-in path? RegQueryValueEx requires a fixed-length buffer, but the default String in VB is an empty variable-length string. Therefore, a fixed-length one is used, then later trimmed. Again, if the request is unsuccessful, the plug-in aborts, and

tells Plus it cannot be used.

Finally, if all the above succeed, the key is closed. Note there is no error-checking here: the plug-in has all the information it needs, and if the key can't be closed, then chances are it was never opened. The code then proceeds to take the fixed-length string, and copy it into the variable-length string. This has to be done one character at a time as VB handles strings differently to C++ (where char* are used). While not perfect, it works, and a faliure hasn't been reported yet.

Now, that was a lot of code to do what is actually a fairly simple task. The rest of the code does more work, yet is shorter. This merely sets up the command strings and any other information that goes with them. Don't worry too much about the fact that they are arrays: the reasons are twofold. One, it makes the other functions easier to write, and two, the code is therefore more easily extensible (with a few minor edits).

Uninitialize comes next, for no reason other than it happens to be next. This is a far simpler function, which simply frees up the WMP resource.

Publishing the Information

The next two functions deal with telling Plus about the commands (no tags are defined). These two are fairly trivial: the variables nCommandCount and nTagCount in GetPublishInfo let Plus know how many times to call GetPublishCommandInfo and GetPublishTagInfo. GetPublishTagInfo is not implemented, and in GetPublishCommandInfo can be seen the reason for using arrays to store the command information.

Parsing the Commands

Again, the arrays make the code easier to write and to extend, however this method does need some slight editing for this to happen. The code is fairly trivial, merely doing string comparisons in an If...Then block. The first block will display a MessageBox if executed, and the second sets up a string for ReceiveNotify.

The Plus built-in sounds can be used with any text, merely by writing some text after the command. The sound command in this plug-in is designed to work in the same way: if sCommandArg (all the text following the command) is empty, a default string is appended to the nCCNotify character and five-character notify code (spaces count).

If the command is recognised, the correct function is performed, and the plug-in returns success. This stops other plugins parsing the command: after all, they probably won't understand it anyway. If the command is not recognised, nothing happens, and the plug-in tells Plus it can pass the command to other plugins.

Receiving the Notification

All the above code has been leading to this. Thankfully, it is a fairly simple procedure. First, the plug-in checks to see if it recognises the notify code. If it does, to plays the sound file specified using WMP. The path to the file was made by appending the correct filename and subfolder to the string data read from the Registry. This way, the plug-in can work no matter where Plus is installed.

To use WMP effectively is fairly simple. First, close the currently open file (if there isn't one, this command does nothing). Then, load the sound file to be played. Once done, play the file. While the code as it stands does not check for errors, if the file cannot be found, then the plug-in proceeds as normal. If an author wishes to notify his users that the file is missing, this would be fairly easy to implement: by and large though, it is not necessary, especially if all the files are in the right places.

Summary

Some of the code in this plug-in may seem complicated, but it is almost exclusively to do with Registry access. As the rest of the code shows, writing a good plug-in takes not a lot of effort. Writing a great plug-in on the other hand, takes a lot of effort, mostly in design work.

This plug-in does not use toasts or Messenger objects, but the following sample plug-in does use DisplayToast.

Toasts galore!

This plug-in is nothing more than a toy to demonstrate the use of DisplayToast, and is therefore very simple.

What the plug-in does

Two commands again. One pops up a toast with an embedded hyperlink, the other one doesn't.

Overview of the code

The plug-in is coded using Visual C++ 6 Professional Edition SP5. The filename is ToastsGalore.cpp.

Resources needed: A .def file to tell the C++ compiler what functions to export. Copy the sample that comes with the MPPlugin sample, and change it to reflect your won plug-in.

Global declarations

There are none in this toy plug-in, but there are two include statements. The first allows the use of various string functions, and the second is the Messenger Plus API definitions file.

DLLMain

This function is a mere formality usually. Just copy and paste this function into your own code.

Initialisation and Uninitialisation

No fancy resources used, so Initialize returns TRUE. There is no Uninitialize.

Publishing the Information

This is only one function compared to VB's 3, but the use of pointers to arrays means that care must be taken to refer to them correctly.

Parsing the Commands

Again it relies on a simple text comparison. The default URL is only used of the length of the sCommandArg buffer is zero.

The DisplayToast function takes four arguments, as described when the APIs were studied in an earlier section. There is a screenshot of the non-URL toast in the package with the source code, called ToastsGalore.jpg.

If you like that display picture, it is from http://www.mess.be/.

Summary

C++ code may not be quite as aesthetically pleasing, but is gets the job done very quickly. The DisplayToast for VB is used in the same way, but it returns a Boolean value.

The Author

Hello, my name is RaceProUK. I have been using Plus since version 1.42, and since November 2003, have been producing plugins for Plus 2. I wrote this guide in March 2004 in response to a fellow Messenger Plus Forums member's request for a plug-in authoring tutorial. While a quick tutorial is a good idea, I decided to expand on the concept and write this guide. It covers the API in both C++ and VB, goes over the MPPlugin sample, and provides some useful code snippets to perform useful tasks.

If you have any feedback, positive or negative, e-mail me at raceprouk@hotmail.com with the subject line 'Plus Plugin Guide' or something similar, or visit the forum at my website, http://www.geocities.com/raceprouk/msn/. I welcome all feedback, as well as code samples. All contributors will be credited.

A bit of small text that tries to stop me getting into trouble: If I have missed any crediting, please let me know. Also, notify me of any copyright infringements, and other legal stuff that I'm doing wrong.

Also, all code samples come with no guarantee.