Version 1.6 – January 24, 2003 – Nathan Yocom

This document is provided as a quick and dirty guide to pGina plugin development. It is by no means complete, free of errors, or guaranteed to be right. – Nate Yocom

Errata – 6/3/2002: *Plugins for pGina >= 1.3 must support UNICODE. The basic steps outlined in this guide still apply, but please refer to the example source code in the plugin SDK for the data types etc.*

Errata –1/23/2002: *pGina 1.6.0 is backward compatible with 1.3+ plugins, however, they will be assumed as "not required" according to the IsRequired() functionality of 1.6+ plugins.*

A Plugin Examination (Step by Step…)

For a basic start, lets examine the DummyPlugin example that is included with the pGina Plugin SDK Version 1.6 and see how it works.

Requirements

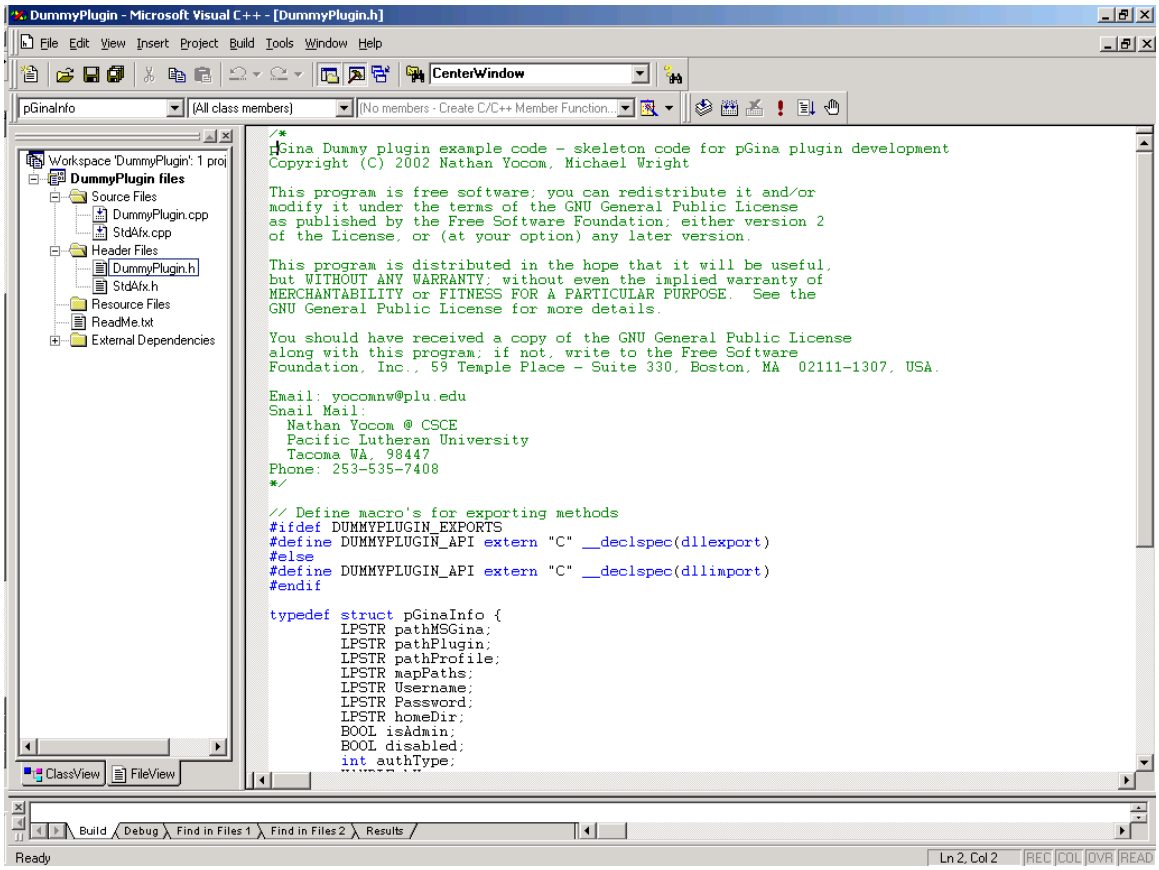
So first thing's first, review the requirements for developing a plugin for pGina.

- Microsoft Visual Studio 6.0 SP5 or newer

And that's it! Theoretically any compiler that can create a DLL with exports should work, but I only have experience with Microsoft's so that is what we will use here.

The Project

Lets open the DummyPlugin.dsw workspace and take a look around.



This should be what you see initially (if not, try double clicking on the DummyPlugin.h in the left panel).

DummyPlugin.h – Necessary Definitions

I am going to assume you have some familiarity with the development environment, so lets dive into the code. Looking first at the DummyPlugin.h file we see some definitions:

```
#define DUMMYPLUGIN_API extern "C" __declspec(dllexport)
```

What this does for us is export all functions that begin with DUMMYPLUGIN_API in the correct manner for pGina. Macros are wonderful aren't they?

Next we come across some important information.

```
typedef struct pGinaInfo {
	LPSTR pathMSGina;
	LPSTR pathPlugin;
	LPSTR pathProfile;
	LPSTR mapPaths;
	LPSTR Username;
	LPSTR Password;
	LPSTR homeDir;
	BOOL isAdmin;
	BOOL disabled;
	int authType;
	HANDLE hUser;
	LPSTR userGroups;
	LPSTR userDescription;
	LPSTR userFullName;
	BOOL allowPassChange;
	LPSTR errorString;
	LPSTR defaultDomain;
	BOOL Reserved3;
	BOOL Reserved4;
} pGinaInfo ;
```

This structure is core to pGina and must be defined in your plugin. We will familiarize ourselves with some of the specific members later. In short, this is the structure that contains all of the data passed between pGina and your plugin.

A little farther down we see the last of the header file in the form of our function declarations. You can define any other functions you want here, but these have to be there in order for the plugin to be valid.

The Meat – UserLogin

Okay, so now we have what we need declared, lets move to the real meat of the plugin, the UserLogin() function. Look in the DummyPlugin.cpp file and locate the function.

Now what does this plugin do by default? Well, the plugin as it stands will authenticate only two users (in addition to users with local accounts). Those users are adminuser and reguser. When we look at the UserLogin() function we can see that the first thing it does is check to see if the passed username is adminuser.

If the username is adminuser, the plugin does some interesting stuff, and here we see the use of some members of the pGinaInfo structure. The first thing it does is set settingsInfo->allowPassChange to a 1. This is equivalent to settingsInfo->allowPassChange = true and when pGina gets control back from this plugin, this member indicates to it that the user should be able to change their password after logging in.

The next thing we see is:

```
	settingsInfo->userDescription = strdup("An …");
```

By placing a string here (strdup mallocs the memory and everything), when pGina logs the user in, it will fill in the User's description field with this string. Similarly, the next line sets the user's fullname field.

The last thing the plugin sets is isAdmin. This flag indicates to pGina that the user should be setup as a local administrator.

The plugin then returns true, so that pGina knows the user is valid, and can process the settings and get things moving… otherwise, the plugin goes on the check and see if the user is named reguser instead, and sets flags in a similar manner.

The other functions simply exhibit default behavior, and as a result we will save examination of them for a more advanced tutorial (anyone want to volunteer?).

A plugin could similarly fill in the errorString member, indicating what went wrong with a FALSE return (pGina will then display that error instead of the typical "Access Denied" message), the defaultDomain member, indicating where user functions should affect (a domain, or if NULL, the local machine) etc.

Review

Writing a plugin:

1. Download and install the pGina Plugin SDK (This document comes with it)
2. The easiest way to get started is to build off of the included DummyPlugin example. This example already has everything defined as it should be, but doesn't have any functionality to speak of (its really just a template).
3. The function you should be most interested in is the UserLogin() function. It is here that you have a chance to do as you will and say yeah or nay (return true of false) to a user logging in.
4. Look at the Plugin Interface Definition, also installed with the SDK – this shows all the different functions a plugin should have, as well as a short explanation of all the pGinaInfo members.

Testing a Plugin

1. The easiest way to test that your plugin exports the correct functions is to use the Configure application provided with pGina. Using this application, run it as if you were going to change pGina settings (Configure pGina) then type the path to the plugin you want to test in the pathPlugin box (DON'T HIT SAVE!) and then hit the Test Plugin button. You will get an error if all of the expected functions are not available etc. Note that this does not guarantee a functioning plugin, just that pGina wont crash when it loads it.
2. Starting with version 1.4 of the SDK, there is now a plugin_tester application. Simply run it, select Load Plugin, point it to your plugin, then use it to simulate logins, hooks, password changes etc. By compiling the testing app in debug mode, and your plugin in debug mode, you should be able to step through your plugin in the interactive debugger.
3. You can also use the Configure app to point pGina to your plugin and reboot (in this case hit save ☺). Keep in mind that if your plugin bombs there is no safe way out. So be prepared with a boot disk with a dummyplugin or something to replace your plugin if things go bad.

Questions Answered

Where should my plugin store its settings?

That is really up to you. However, we recommend keeping your settings in the registry in the HKLM\Software\pGina\<yourpluginname> key. In this way it is easy to identify your settings as pGina related, but you don't risk butting heads as pGina matures.

I am trying to link to an external library and am getting errors that indicate DllMain() is already defined. What do I do?

DllMain() is not required for your plugin to function. As a result, you have two options, you can remove it from your code (as long as you don't need to do anything there) or you can use the /FORCE:MULTIPLE compiler setting to make the compiler ignore the linked libraries DllMain. The results of either method and

its impact on your ability to do things is mitigated by the libraries you are linking and what type of things you are doing (i.e. beyond the scope of this document).

How can I add users to a group or groups when they login?
In order to have users added to groups by a plugin you have to have the groups already created. If that is the case, then all the plugin has to do is place a list of the groups the user should be added to (separated by semi-colons) in settingsInfo->userGroups prior to returning control from UserLogin(). For example, if you wanted the user to be added to "GroupA" and "GroupB" you could simply do:

```
SettingsInfo->userGroups = strdup("GroupA;GroupB");
```

Do I have to provide fullname and descriptions for all users?
Nope, if you don't specify then pGina will leave them blank (NULL).

Where do I have to implement error notifications etc?
When your plugin encounters an error that should be provided to the user, place the text of such in the errorString member of pGinaInfo. Currently this is only examined (in 1.6.0) at login time, but in future revisions this will always be checked upon return from the plugin. In the meantime, you should notify users however you feel is necessary anywhere a problem may occur. pGina also has an event viewer topic, which you may log to – look at the logOut() function in pGina.c of the pGina code to see how this is done.

What is the HANDLE hUser member of pGinaInfo for?
This is a copy of the users security context (handle). This is provided in case a plugin should need it for anything in-particular. The example that jumps to mind is drive mapping. Drive mapping requires that you make a call to ImpersonateUser() first, which requires a copy of the users security context. We felt giving it to you was easier than making you get it, or request it yourself.

Other questions?
Email me, nate@yocom.org, or join one or all of the pGina mailing lists – links to which are available on the website at http://pgina.cs.plu.edu/.