Maya API Wizard

User Manual

Callum James

Contents

1. Introduction

Welcome to the Maya API Wizard. This user manual will guide you through simple steps to not only set up and build the wizard from source but also how to get you generating framework for Maya plugins.

The purpose of this plugin is to make Maya API and Plugin development simpler to set up and get started with. To do it all manually can be messy affair and it is very easy to make small simple mistakes when remembering which functions needs to be overridden and which includes need to be added to your files. The Maya API Wizard takes care of all of this for you. By following the Wizard through and filling in the required fields on each page, you can start plugin development on an already set up environment with inheritance, virtual overriding and even attribute creation handled for you.

Once created, you can make and build the libraries for each plugin and load it straight into Maya. However this wont actually do anything as their is no functional code generated. This is all down to you. Gaps are left so that you can in any direction you want to develop the next greatest plugin. The boring stuff just gets done for you! Now thats service!

Currently, the Wizard runs on both Linux and MacOSX but not Windows. Consequently, the Wizard currently only generates framework that will build and compile on Linux and MacOSX and not on Windows. These are the only limitations that are currently imposed on the Wizard.

2. Building the Wizard

If you have the Wizard then you will have access to its full source code. Therefore before you can run it and start developing Maya plugins, you will need to build it. To do this is fairly straight forward. However there are a couple of things you need to make sure you have before attempting the build:

- Qt 5
- Boost (filesystem)
- The node graph library built for this wizard

Once you have made sure you have these things, we can begin. The Wizard will currently only build on MacOSX and Linux.

In a terminal, navigate to the directory where the source is stored. Within this directory, you should see a MayaAPIWizard.pro file. This can either be opened in Qt Creator if you have this IDE, or built straight through using qmake.

Using Qt Creator

Open the .pro file within Qt Creator and configure it properly, making sure you are using a 64 bit compiler and libraries. Then, if need be, change the values in the .pro file to match where your installs of boost are. Once this has done, go to Build->Run qmake.

Once qmake has been run, you will notice there is a Makefile within the root directory. Either click Build->Build All within Qt Creator or run make from the command line. The Wizard should successfully build. If all has gone well, you are all good to go!

Using qmake

To use qmake, first make any changes within the .pro file you need to using a text editor of your choice, just as above. Once this is done, from the command line within the MayaAPIWizard root directory, run qmake. A Makefile will now appear in the directory. The simply run make. The wizard should now be built and ready to go.

3. Using the Wizard

Once built from source, the Wizard will be run-able. Once past the introduction page that simply explains how the Wizard will work, you will find yourself on the plugin type selection page. On this page, you need to fill in a couple of required fields.

The first of these is the directory within which your Maya exists. This can be found by clicking the browse button and locating the right folder. Please make sure this is correct as when attempting to move on from this page, the Wizard will conduct a test to make sure it is a valid Maya directory. In this test, it looks for libOpenMaya and MGlobal.h. If it cannot find either of these, it will report an error.

You also need to enter a Vendor name, plugin version (this will also be used for naming created directories) and the require Maya API version. The default of this last one is Any but can be changed to what you desire. All of these MUST be filled in before continuing.

These are all generic attributes common to all Maya plugin types and so will need to be filled in for any Maya plugin that you create.

The final thing on this page is the plugin type selection. From this page, there are 6 possible selections. There are however more possibilities within the Node selection. Depending on which plugin type you choose will change your path through the wizard.

3A. Supported Plugin Types

As of the current iteration of the Wizard, there are 17 supported plugins that can be generated using this wizard. Some of these are variations on others but all require their own unique set up. The current supported plugin types are:

- Mel Command
- File Translator
- Node
    - Depend Node
    - Locator Node
    - Deformer Node
    - Manipulator Container Node
    - IK Solver Node
    - Hardware Shader Node (different from Hardware Shader)
    - Transform Node
    - Object Set Node
    - Image Plane Node
    - Particle Attribute Mapper Node
    - Constraint Node
    - Manipulator Node
- Software Shader
- Hardware Shader
- Dynamic Emitter

To select the plugin you wish to create a template for, simply select the one from this list, fill in the required fields and click Next. This is how you will navigate through the rest of the Wizard. On each page, there will be certain fields that must be filled in and check boxes and selections that are optional. Once you are happy with your selections, click next to simply move on. There are a few steps to complete to generate framework, but once these have been completed, a fully structured plugin template will be available for you to work with.

3B. Linking Libraries

Independent of which plugin type you have chosen to create, you will at some stage be presented with the Link Libraries page. Here you have the option of automatically linking in certain libraries when the framework is generated. By default, the Foundation and OpenMaya library are always required and so are included by default. The Wizard will also automatically include and lock any libraries that are required by your chosen plugin type. The others you are free to include or not if you wish. Included some may help with future development of your plugin so you don't forget to link them in when you need them. However, any library option that is not checked and locked on this page are not vital for your chosen plugin type and so can be left excluded or included if you wish.

To include the other libraries, simply check the ones you want included, and then click next!

3C. Generation

The final page before generation occur will be the Generate page. Here it will present you with a final few options that need to be filled out and decided upon. The first of these is if you wish to generate some kind of Makefile generator. The first option available is a Qt project, which will create a .pro file that allows project development within Qt Creator and compilation with qmake and make. The second option is the CMake file (CMakeLists.txt). If you select this option, the Wizard will create a CMakeLists.txt that will compile straight out of the hat and an extra build directory for you to build into. If you wish, you can select neither of these, both or them or neither.

The next option is where to write the files to. This is the only required field on the page and determines where the framework files will be written to.

Finally, you can choose to either create a directory or not. If you choose to create a new directory, all of your source files will be neatly organised and contained within a single folder. If you choose not to, all files and folders will simply be written to the specified location. If you do choose to create a new folder, the plugin version will be used to help identify different iterations of a similar plugin made with the same name.

When you are happy with all of these options, click Generate, and assuming all goes well, which it will, your plugin framework will be generated and written out to where you told it to. From this you then start plugin development or if you wish to test it works, compile it straight away and load into Maya.

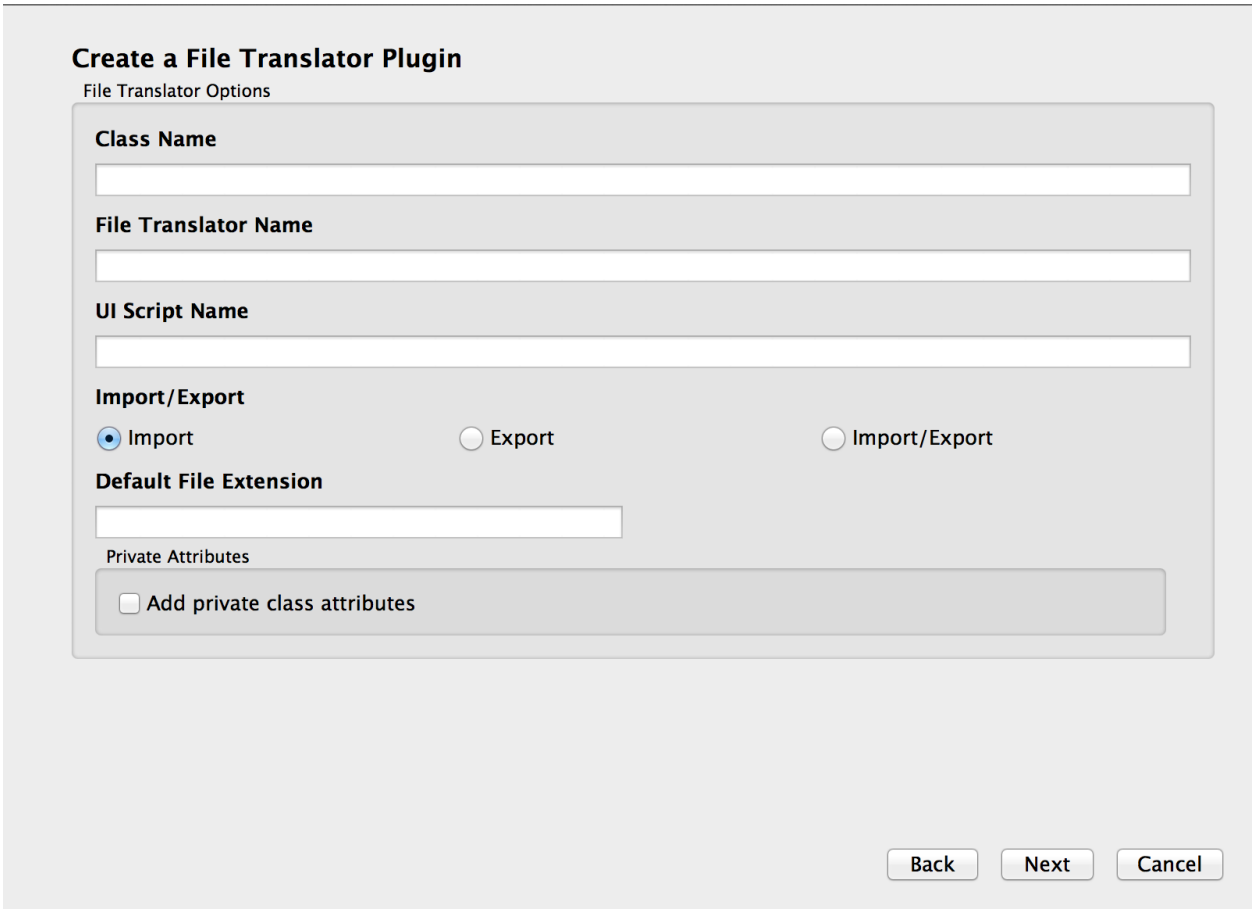
4. Plugin Specific Pages

In addition to the standard pages that you will always use whilst generating a plugin, the page for each type of plugin is different specific to that plugin. Each one will contain different options and different required fields for you to fill in. As a general rule, any text entry locations will need to have a value for you to proceed in the Wizard and any check boxes are normally optional.

On each specific page, there are simple instructions explaining what each fields or option means, allowing to set up your plugin as fully as you can straight from the off.

For example, below is the page for a File Translator plugin.

On this page, there are four required fields. The class name must be filled in with a name for the class itself. Then the File Translator name also needs to be filled in. This is the name used to register the plugin with Maya and the one that appears in the import/export bar. It CANNOT be the same as the class name otherwise Maya has issues. Next you must give a name to the UI script. This is the name the file itself will be called and how it will be referenced from the plugin. This UI script is used within the import/export window to allow the user to select options for the process. The final required field is the Default File Extension. This is a file



extension that the translator will by default look for when scanning for files.

All other options are optional. The import and export selection will simply set the File Translator to either import, export files or both. The final check box Add private class attributes will give you the option to add private members that you know you will need within this plugin. The way that these are added are discussed in section 5 of this manual.

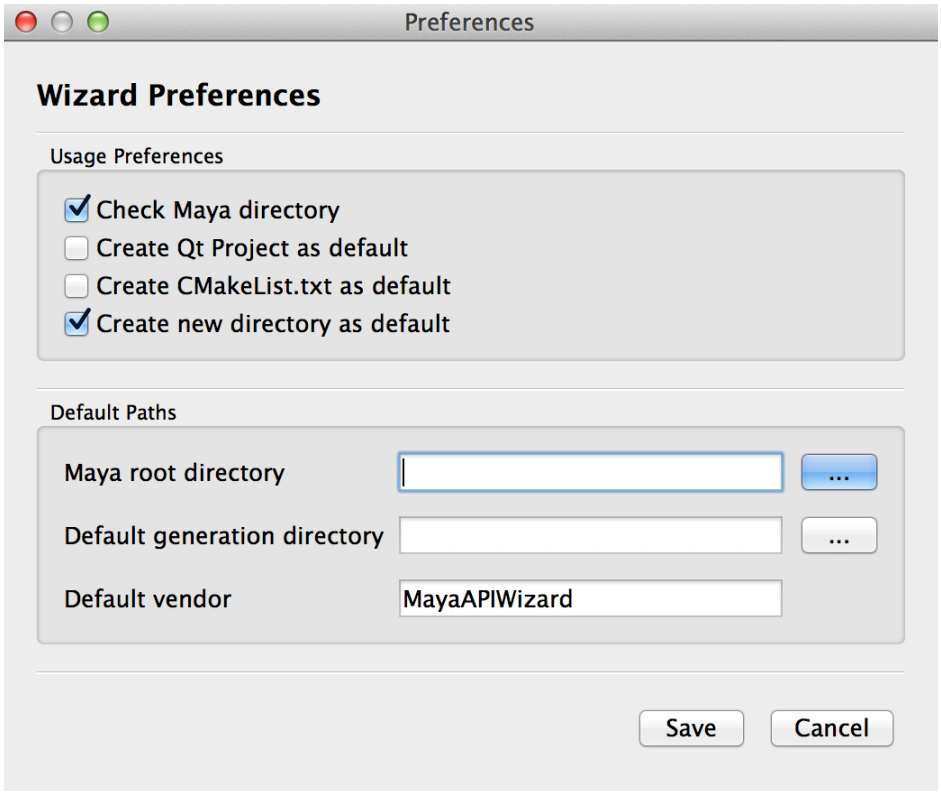
Once all of these have been filled in, you can click next to proceed with the Wizard. Every other page will follow the same sort of format and try to make generation as simple as possible for you.

5. Preferences

There are a few preferences that can be modified and set for the Wizard. To access the Preference window, go to:

Linux: Edit->Preferences
MacOSX: MayaAPIWizard->Preferences

The following window will appear:



The window you see here show the default preference value when you first build the wizard or run it for the first time.

Check Maya directory, if checked, will cause the Wizard to check the supplied maya root directory for validity. If unchecked, this analysis will not be done.

Create Qt Project as default, if checked, will set the option to create a Qt project file with the generated source as true by default.

Create CMakeLists.txt as default, if checked, will set the option to create a CMake file with the generated source as true by default.

Create new directory as default, if checked, will set the option to create a new directory for the generated source files to true when reaching the generator page.

There are three default paths you can set. The first is the maya root directory which is where your Maya install is located. This can be left blank, although this will require you to

browse for the directory each time. To browse for this directory, click the … button next to the text input and find the right folder.

The next is the default generation directory. This is where the wizard will set to write new source files and folders to unless changed. Again this can also be left blank if need be, although the path will then always need to be entered upon generation. To browse for this default directory, click the … button next to the text input.

The final value is the default Vendor name. This is defaulted to MayaAPIWizard but can be changed to anything you wish, again left blank if you want. This is simply the default value the wizard will place into the vendor option when selecting a plugin type.

To save all changes to file, click save. The wizard will automatically pick up on the new changes if moving on to the next page. When moving back, the entered values will be kept. If you navigate back past a page, and then move forward again, any default values applicable will then be entered.

To cancel all changes made, simply click cancel.

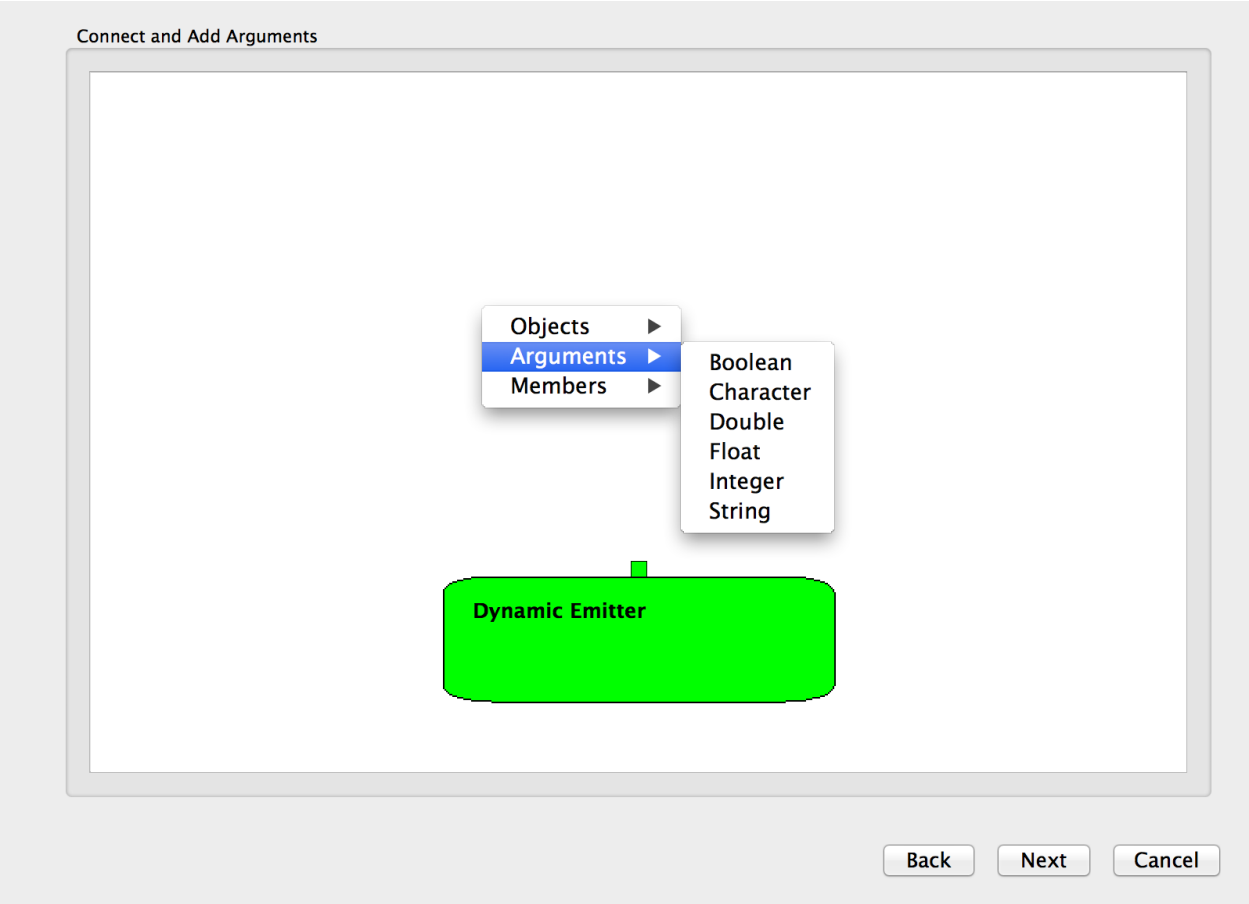
6. The Node Graph System

As discussed at the end section 4, the Wizard gives you the option to add extra members/arguments/objects dependant on the type of plugin you are creating. So for example if you are creating a Mel Command you can add command line arguments as well as private class members. This option is triggered if the Add Members checkbox is checked on each plugin specific page before continuing with the Wizard process. If this option is selected, you will be presented with a page similar to below:

This a custom node based system developed to make the creation of attributes simple and fast. All nodes are moveable by right clicking on them dragging. You can zoom in and out

using the mouse scroll wheel.

The node you see above is your target node and is not deletable. It will be garnished with the type of plugin you have chosen to generate. This is the node to which all objects and members must be connected to.

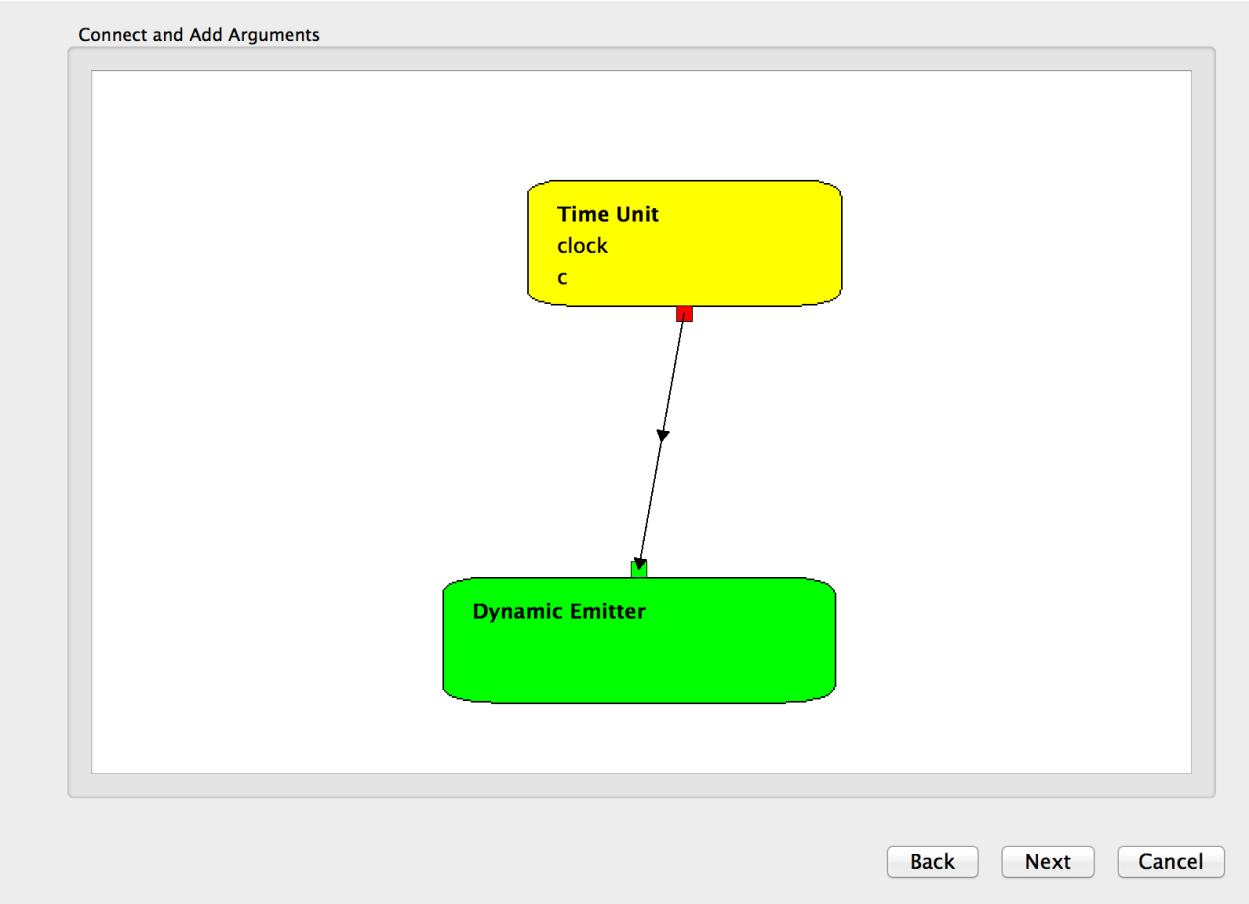


There are three different types of nodes you can create, Objects, Arguments and Members. To start creating, with your mouse over the scene, press the spacebar and a menu will pop up at your mouse cursor.

Select the node you wish to create and it will appear within the scene. Move it to where you are happy. Now we need to edit it. To do this, double click on it to bring up the edit window. Here you need to enter a name (10 characters maximum) and a short name (4 characters maximum). The short character is used mainly within the command line arguments for Mel Commands. None of the names or short names can be the same. To close the editor window, just hit Return, Enter or Escape.

Once edited, we need to make sure they will be read by the wizard. If you have created a member (cyan), you do not need to do anything else. They will all be read and created as private members of your class.

If you have created an object or an argument, you will need to click and drag their outbound socket (red) and connect it to the green inbound socket on the target node. Only connected nodes will be read!



To delete nodes, simply select it and hit either delete or backspace!

When you are happy with all of the nodes connected and members created, continue as normal with the Wizard process. These extra attributes will be created within your plugin class. If you have created invalid members (such as arguments for a node plugin) they will just be ignored.

Any objects you create will not only be declared in your class but also very simply set up with their right Maya data type within the cpp of your plugin class. You however will need to do anything with them and add them to your plugin.

7. Building Your Generated Plugins

Once you have created your framework for your plugin, you will want to test that it builds. Depending on how you have selected it to be generated will change how to do this.

If you decided to not generate either a Qt project file or a CMake file, first you are going to have set up a build environment for the project. How you do this is up to you and will change how you go about building it. We will assume that if you chose this option you have a preferred build method and so know what you are doing.

If you chose to create a Qt project file, getting your plugin built is easy. Either open it in Qt Creator, edit it, develop it and build it from there, or just run qmake on the .pro from your command line and then run make. The Qt project file will allow you to easily develop and further the plugin after the basic framework has been created.

If you chose to create a CMake file (CMakeLists.txt) then you first need to make sure that you have CMake installed on your machine (version 2.6 or later). Once you have clarified that this is the case, from your command line, enter the build folder located within your plugin directory (this folder will only be generated if a CMake file is generated). Once in the build folder, run the following:

```
cmake ..
make
```

This will generate a Makefile and then build your library. Once your libraries have been built, copy it and any accompanying scripts (found in the scripts folder) to your Maya plugins and scripts directories respectively. These will then be available for Maya to load in.

Its as easy as that.

8. What Does it Actually Generate?

When you click generate on the final page, what does this Wizard actually create for you? It will generate basic framework for your chosen plugin type and organise it for you within easy to follow folders.

First we will start by examining the directory structure created by the Wizard. The generated structure is the same for every plugin type. When we discuss the root directory, this is simply the directory into which the folders and source will be written. This may be the folder you chose, or the newly created plugin directory if you chose to create a new directory. Within this root folder, the Wizard will create 5 directories:

- root directory
  - include - all header files
  - src - all source files
  - obj - directory to build obj files into for Qt
  - moc - directory for moc files for Qt
  - scripts - all scripts

If you choose CMake build, an extra folder, build will be created for you to build your project into.

The Qt project and/or CMake file will be created into the root directory.

Per plugin, typically, the WIzard will generate one header file. The only difference is the Hardware Shader which requires two for two classes. This header file is the header for the plugin class and contains the class declaration, functions and members. It will be named the same as the class with Plugin appended.

In the src folder, typically two cpp files will be generated (again this differs with the Hardware Shader where there are three). These are the cpp source for the plugin that includes the header just mentioned. The functions in this file will be incomplete, waiting for you to fill in the gaps. These functions however will be either minimum requirements for the plugin type or any extra ones you have chosen to create. The other source file is the main.cpp. This is where the plugin is initialised and deinitialised. The two functions in here will always be filled out and complete for you and will simply allow the plugin to be loaded straight into Maya.

If the plugin requires scripts to accompany it (which is the majority) these will created in the scripts folder. These will either be named within the Maya convention for the plugin type or if you have been asked to specify a script name, it will be called and registered with this name.

This is the basis of what this Wizard will generate for you for the different plugin types.