CloudCompare Coding rules

=======================

## Naming

Names in CloudCompare should be as descriptive as possible, without abbreviations, apart for very clear or common ones (such as `fw` instead of `forward`, etc.). Most of variable names should begin with a lower case letter. If the name is composed of multiple words, the first letter of each word should be in upper case (apart for the first one of course).

Example: `numberOfPoints`, `ptsCount` (or even `ptsNum` for the laziest ;-)

### Specific cases

- static variables: should always begin with prefix `s\_` (in lower case – like `s\_defaultFilename`

- static methods: should always begin with a upper case letter (like `InitGLEW`)

- classes: should always begin with prefix `cc` (in lower case – like `ccConsole`)

- enumerators:

- all letters in upper case

- should always begin with prefix `CC\_`

- words are separated by underscore (like `CC\_OBJECT\_FLAG`)

- macros: begin with prefix `MACRO\_` followed by a standard method name (like `MACRO\_SkipUnselected`)

- const variables:

- all letters in upper case

- words are separated by underscore (like `NORMALS\_QUANTIZE\_LEVEL`)

- `macro const` (`#define`): should be avoided; same syntax as const

### Files

- File naming follows the same rule as most CloudCompare elements (first letter in lower case, etc.)

- Each class should be saved alone in a header + source file couple. Exceptionally, very small classes that are used by a single class may be saved along with this class. The header + source filename should be the same as the main class.

Example: `ccConsole` saved in `ccConsole.h` and `ccConsole.cpp`

- Filenames shouldn’t contain any space character. Use underscore instead.

- All data-related classes (data models, database, etc) should be saved in `db` directory.

- Images (icons) should all be saved in the `images` directory (or one of its subdirectories).

- GUI templates (mainly `.ui` Qt files) should be saved in the `ui\_templates` directory

### Tabs and indentation

- Indentation is expected to be made in `Tabs` only, each of size `4`.

### Summary

Element | Example

------- | -------

Class | `ccMyClass`

File | `ccMyClass.h` and `ccMyClass.cpp`

Attribute/variable | `myAttribute`

Static attribute/variable | `s\_myAttribute`

Method | `getMethod()`

Static method | `GetMethod()`

Structure | `myStruct`

Enumerator | `CC\_MY\_ENUMERATOR`

Macro | `MACRO\_myMethod`

Const variables | `MY\_CONSTANT`

Const (`#define`) | `MY\_CONSTANT`

## Unix compliance

For avoiding incompatible syntax with Unix environments, the following rules must be respected:

¬ use only "/" for include paths.

Example: `include "../db/ccPointCloud.h"`

## File headers

Any new source file (`.h`, `.cpp`, etc.) integrated to any CloudCompare module (CCLib, qCC, etc.) must present the official header.

Here is the official header for LGPL modules (CCLib, etc.):

```

//##########################################################################

//# #

//# MODULE NAME #

//# #

//# This program is free software; you can redistribute it and/or modify #

//# it under the terms of the GNU Library General Public License as #

//# published by the Free Software Foundation; version 2 of the License. #

//# #

//# This program is distributed in the hope that it will be useful, #

//# but WITHOUT ANY WARRANTY; without even the implied warranty of #

//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the #

//# GNU General Public License for more details. #

//# #

//# COPYRIGHT: XXX #

//# #

//##########################################################################

```

And for official header for GPL modules (qCC, etc.):

```

//##########################################################################

//# #

//# MODULE NAME #

//# #

//# This program is free software; you can redistribute it and/or modify #

//# it under the terms of the GNU General Public License as published by #

//# the Free Software Foundation; version 2 of the License. #

//# #

//# This program is distributed in the hope that it will be useful, #

//# but WITHOUT ANY WARRANTY; without even the implied warranty of #

//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the #

//# GNU General Public License for more details. #

//# #

//# COPYRIGHT: XXX #

//# #

//##########################################################################

```

Designing a new qCC plugin

==========================

## Introduction

Designing a new plugin is an easy way to extend qCC (CloudCompare) functionalities, without the pain of having to modify its core and do all the connections.

One can easily design a new function, that may be applied on one or several entities currently loaded in CloudCompare. Moreover, the plugin can display its own dialog.

A dummy plugin structure (the sources and the corresponding Code::Blocks project) is provided as a template.

## First steps

Here are the first mandatory steps to create a new plugin. In fact these following setps are basically meant to build up a working Code::Blocks project for your new plugin based on the qDummyPlugin template.

The different plugins projects are located in the `plugins` folder. You should see a folder named `qDummyPlugin` inside.

1. Simply “copy” and paste the `qDummyPlugin` folder it in the same directory (trunk\plugins)

2. You should see now a new folder (“copy of qDummyPlugin” or “copie de qDummyPLugin” in French)

3. Rename this directory with you own plugin name (for instance “qMyPlygin” for this tutorial).

4. browse to this directory

5. you should see the following files inside:

- `qDummyPlugin.h` & `qDummyPlugin.cpp`: the source files

- `qDummyPlugin.qrc`: a Qt resource file (for icons, etc.)

- `icon.png`: a fake icon file

- `CMakeLists.txt`: CMake configuration script

6. Rename all the `qDummyPlugin.\*` files with you own project name

- `qDummyPlugin.h` => `qMyPlugin.h`

- `qDummyPlugin.cpp` => `qMyPlugin.cpp`

7. Edit the `CMakeLists.txt`:

- Replace all occurrences of `DUMMY` with your plugin name (don’t forget any or conflicts may occur with existing CMake variables

- If your plugin relies on additional libraries, you should also add them here. See for instance the equivalent files for the qHPR or qPCV plugins.

### Modifying the sources

You can now begin with the real work: implementing the plugin action. There are some modifications that have to be done first however.

#### Header file

Open the header file (`qMyPlugin.h`).

1. at the top of the file you should see first a standard `CloudCompare` header. You can change inside the plugin name (`qDummy` => `qMyPlugin`) and the copyright owner.

2. below this header, we have a standard C++ class declaration.

- you should modify the macro word `Q\_DUMMY\_PLUGIN\_HEADER` with your own (for instance: `Q\_MY\_PLUGIN\_ HEADER`). Do it on both lines.

- you should also update the class description (Doxygen style)

- and eventually rename the class itself (`qDummyPlugin` `qMyPlugin`)

3. This is all that has to be done for the header file.

#### Source file

Open the source file (`qMyPlugin.cpp`).

1. Same thing: you may update the header (plugin name and copyright owner).

2. then, read carefully all the comments (there are basically the same information as below):

- replace all occurrences of qDummyPlugin by your plugin class name (`qDummyPlugin` `qMyPlugin`). You may use the `replacing tool` to do this (Menu `Search > Replace` or `CTRL+R`). Make sure the `Whole word` and `Match case` checkboxes are checked, and then click on the `Replace` button, and eventually on the `All` button.

- now only two mandatory steps remain:

- update the `getDescription` method (especially, you should replace the `Dummy Plugin` string by your plugin name and the `Dummy Action` string by a short description of your plugin action).

- put your code in the `doAction` method (between the two `/\*\*\* HERE STARTS THE MAIN PLUGIN ACTION \*\*\*/` delimiters).

Whenever the user clicks on your plugin icon, CloudCompare will call this method.

- Optionally:

- You can access most of CloudCompare resources through the `m\_app` member (an interface to the main application: data base, main window, 3D view(s), etc.).

- To determine which entities were selected when the user clicked on the icon(s) or if the icon should be enabled or not, you should add custom code to the `onNewSelection` method (this method is called whenever the selection changes).

Using CCLIb and CloudCompare database/algorithms

All algorithms (in CCLib) and 3D entities (in CCLib, qCC\_db, qCC\_io and qCC\_gl) are accessible inside the plugin. Check the doxygen documentation of those projects for more information.

- [CCLib doxygen documentation](http://www.cloudcompare.org/doc/CCLib/html/index.html)

- [qCC doxygen documentation](http://www.cloudcompare.org/doc/qCC/html/index.html)

Once again, the other plugin projects are a good source of hints, as the CloudCompare project itself.