Richlinien zur Programmerstellung von ProMalDes in c++

by Daniel Bachmann

Grundsätzliches Ziel dieser Richtlinien ist es, einheitlichen und übersichtlichen Quellcode zu erzeugen, der auch von anderen einigermaßen nachvollzogen werden kann.

Grundsätzlich gilt die **englische** Sprache, d.h. für Kommentare und Namensgebungen.

Namensgebung:

Modulnamen:

Sys (Systemmodul)
Geo (Geometriemodul; Untermodul von Sys)
Fpl (Versagenswahrscheinlichkeiten
Hyd (Hydraulikmodul)
Madm (Multiattribute Decision-Modul)
Dam (Schadensmodul)
Alt (Alternativemodul)
Cost (Kostenmodul)

Files der Klassen/Klassenname (.h;.cpp)/Strukturen/Enumertaions/ Namespaces:

Jede **Klasse** hat einen .h-file und einen .cpp-file; die Namen der files entsprechen dem Klassennamen: Trennung durch underline; neue Wörter groß schreiben. Rein **virtuelle Klassen** beginnen mit einem underline.

Bsp. Klassenname:

```
class Modulname_Class_A
{
      Klassendeklaration
};
```

Aufbau **Filename**: Modulname_Class_A.h; Modulname_Class_A.cpp; Modulname_Class_A.ui etc.
Bsp. Filenamen:
Hyd_Hydraulic_System.h
_Hyd_Model.h (rein virtuell)

Dialoge enden zusätzlich auf Dia; **Widgets** auf Wid.
Bsp. Dialog:
Sys_Output_Lofile_Dia.h

Strukturen und **Enumeration** werden bezeichnet mit _modulname_name
Bsp. _hyd_data_structure oder _hyd_enum

Namespaces werden bezeichnet mit modulname_name
Bsp. hyd_namespace

Methoden (methods):

Trennung durch underline; neue Wörter klein schreiben.

```
void my_method(const int test) etc.
```

Bsp. Methodenname:

```
void check_parameters_found(void);
```

Eigenschaften (members):

Trennung durch underline; neue Wörter klein schreiben

```
bool my_test_eigenschaft;
```

Bsp. Eigenschaftname:

```
bool found_angle;
```

Kommentare/doxygen:

Grundsätzlich viel kommentieren; der Code soll auch für andere verständlich sein!
/// **(3x slash)** bedeuted eine **Kurzbeschreibung** für die manual-Erstellung in doxygen
// **(2x slash)** normaler Kommentar; wird in doxygen **nicht** berücksichtigt
Die Klasse, jede Methode und jede Eigenschaft soll im *.h-file durch ///-Kommentare mit einer Kurzbeschreibung (englisch)* versehen werden. Dies erfolgt jeweils eine Zeile über der Deklaration.
Eine detailierte Beschreibung erfolgt durch:
/**
Comment….
*/
Dies sollte nach der Kurzbeschreibung der Klasse und vor der Klassendeklaration (.h-file) erfolgen. Mit \see (anderer Klassenname) kann auf andere Klassen verwiesen werden.
Eine Detailbeschreibung der Methoden findet im .cpp-file in der Methode am Anfang statt.

Alle Kommentare die nicht im Manual erscheinen sollen, wie Kommentare in den Methoden (cpp-file) sind //-Kommentare.

Deklaration, Definition und Initialisierung:

Methoden und Eigenschaften einer Klasse werden im .**h-file deklariert**.

Dabei soll der Lesbarkeit wegen auch ein **void-Übergabeparameter ausformuliert** werden.
Bsp. Void-Übergabe:
`void my_function(void)` nicht `void my_function()`.

Außerdem sollten Variablen nach Möglichkeit **konstant übergeben** werden; falls diese in der Methode geändert werden sollen, dann muss mit Zeigern (Pointer) gearbeitet werden.
Bsp. Const-Übergabe:
`void my_function(const int test)` nicht `void my_function(int test))`.

Jede **deklarierte Methode** wird im **.cpp-file definiert**, nach Möglichkeit in Reihenfolge der Deklaration in der Klasse. Der Kommentar über der Definition im .cpp-file soll gleich dem Kommentar im .h-file sein. Wichtig hier nur //-Kommentar, da die Beschreibung sonst zweifach im Manual aufgeführt!

Jede **deklarierte Eigenschaft** muss im **Constructor initialisert** werden; auch Zeiger sollen =NULL gesetzt werden. Dynamisch allokierte Eigenschaften sind spätestens im Destructor zu zerstören. Danach müssen die Zeiger wieder =NULL gesetzt werden.

Beachte die **Klammersetzung, Kommentierung und Einrückung** in den nachfolgen Beispielfiles.
Grundsätzlich gilt besser mehr Zeilen-Code, wenn es die Lesbarkeit verbessert.
Bsp. Klammerung:

```
if(xy==true){
      mache_z();
}
```

nicht

```
if(xy) mache_z();
```

Benutze in der Definition den **this-> Zeiger**; dadurch wird deutlich, dass es sich um eine Eigenschaft/Methode der Klasse handelt. Außerdem kann durch das aufgehende Menü (Visual Studio) die entsprechende Eigenschaft/Methode leicht herausgesucht werden.

Beispiel Aufbau .h-file:

```
#pragma once
#ifndef HYD_PARSE_FP_H (entspricht Klassenname in Großbuchstaben)
#define HYD_PARSE_FP_H

(Einbinden der header-files)
//hyd_system class
#include "_Hyd_Parse_IO.h"


(Kurzbeschreibung class)
///Class for…
(Detailbeschreibung class)
/**
Class for… und mehr text
(Verweis suf andere Klassen)
\see Class_Xv, Class_Xc
*/
 (Start Klassendeklaration)
class Hyd_Parse_FP
{
public:
      ///Default constructor (Kurzbeschreibung)
      Hyd_Parse_FP(void);
      ///Default destructor (Kurzbeschreibung)
      ~Hyd_Parse_FP(void);

      //members (Kommentar wegen der Lesbarkeit)

      //method (Kommentar wegen der Lesbarkeit)
      (Kurzbeschreibung methods)
      ///Parse for the global floodplainmodel keywords
      void parse_floodplainmodel_params(const int fp_index);

private:
      //members (Kommentar wegen der Lesbarkeit)
      (Kurzbeschreibung member)
      ///Container for all global floodplainmodel parameters;
      Hyd_Param_FP fp_params;

      //methods (Kommentar wegen der Lesbarkeit)
      (Kurzbeschreibung methods)
      ///Parse for general settings
      void parse_general(EnumKeyword Key, word Command);

};
(Ende Klassendeklaration)
#endif
```

Beispiel Aufbau .cpp-file:

```
(In der Regel wird nur der eigene .h-file eingebuden)
#include "Hyd_Parse_FP.h"

//constructor(Kommentar wegen der Lesbarkeit)
Hyd_Parse_FP::Hyd_Parse_FP(void){
      (Initialisierung der Eigenschaften)
      this->found_floodplainfile_name=false;
}
//destructor(Kommentar wegen der Lesbarkeit)
Hyd_Parse_FP::~Hyd_Parse_FP(void){
      (Aufräumen)
}
//___________________(Kommentar wegen der Lesbarkeit)
//public(Kommentar wegen der Lesbarkeit; public methods)
//Parse for the global floodplainmodel keywords (Kommentar wie .h-file nur
//)
void Hyd_Parse_FP::parse_floodplainmodel_params(const int fp_index){
(Detailbeschreibung method)
/**
Method for… und mehr text
(Verweis auf andere method)
\see method_Xv, method_Xc
*/

      (Methodendefinition)

      if(xy==true){
            this->mach_test();
      }
      for(int i=0; i< test2; i++){
            this->test1=this->test1+1;
      }
}
//_________________(Kommentar wegen der Lesbarkeit)
//private(Kommentar wegen der Lesbarkeit; private methods)
//Parse for general settings
void Hyd_Parse_FP::parse_general(EnumKeyword Key, word Command){
      (Methodendefinition)
}
```