2/25/2010

MITK Style Guide

Beautiful Code

dkfz.
GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

Name Convention

- Using case change to indicate separate words

Imagefilter
Pixeltype
Datastorage

ImageFilter
PixelType
DataStorage

- Underscores are not used

Image_Filter
_Node

- Variable names should convey the meaning behind the code

```
BoundingBox::Pointer result = BoundingBox::New();
```

```
BoundingBox::Pointer boundingBox = BoundingBox::New();
```

Name Convention

- Names are generally spelled out

```
mitk::DataTreeNode* n;
```

```
mitk::DataTreeNode* node;
```

- Abbreviation are allowable when in common use

```
ROI
```

Naming Classes

- Classes are named beginning with a capital letter
- Classes are named according to the following general rule:

class name = <algorithm><input><concept>

Examples of concepts:

Container	A container of objects such as points or images	*VectorContainer*
Factory	Object factories are used to create instances	*DataTreeNodeFactory*
Filter	A class that participates in the data processing pipeline	*AddImageFilter*
Mapper	Transform data from one form into another	*Contour2DMapper*
Reader	A class that reads a single data object	*VtkSurfaceReader*

Naming Files

- Files should have the same name as the class, with an mitk
  Qt specific MITK classes with an Qmitk prepended.

mitkDataStorage
QmitkDataStorageComboBox

- Header files ends with an .h
- Implementation files with an .cpp or .txx for a template class

Naming Methods and Functions

- Functions and methods are named beginning with a capital letter
- Referring to class methods in code, an explicit this-> pointer should be used

```
mitk::DataStorage::SetOfObjects::ConstPointer all = GetAll();
```

```
mitk::DataStorage::SetOfObjects::ConstPointer all = this->GetAll();
```

Naming Signal/Slots

Slots

- Slots are named according to the following general rule:

On[variable name who send the signal][signal]();

```
connect( loadImagePushButton, SIGNAL( clicked( bool ) ),
         SLOT( OnLoadImagePushButtonClicked( bool ) ) );

void mitk::Image::OnLoadImagePushButtonClicked( bool )
{
  ... Do something ...
 }
```

Signals

- Signals are named according to the following general rule

Signal[MethodName](); e.g. emit SignalPointListChanged();

Naming Variables

Class Data Members

- Class data member are prepended with m_

```
m_Volumes
m_OffsetTable
m_ImageMask
```

- Except of QT class Data Members, those begins in lowercase.

```
loadImageButton
closeImageAction
```

Local Variables

- Local variables begin in lowercase

```
offset
data
slicesIt
```

Qt Variables

- GUI variables ends with name of used QT tool.

```
QPushButton*   loadImagePushButton;
QAction*       closeImageAction;
QCheckBox*     hideImageCheckBox;
QRadioButton*  binaryImageRadioButton;
```

Naming Variables

Typedefs

- Typedef names end in the word Type

```
typedef TPixel                                    PixelType;
typedef itk::Image< TPixel, VImageDimension >     ImageType;
typedef std::list<mitk::Image::Pointer>           ImageListType;
```

Pointers

Declaration of Pointers

- Position of * pointers are connected with declaration type

```
int *counter;
```

```
int* counter;
```

- Analog to references:

```
int &counter;
```

```
int& counter;
```

Namespace

- MITK classes should be in namespace mitk

```
mitk::Image::Pointer mitk::ImageGenerator::MakeImage()
{
  mitk::Image::Pointer image = mitk::Image::New();
  mitk::ImageDecorator::Pointer decorator = mitk::ImageDecorator::New();
  decorator ->Decorate( image );
  return image;
}
```

```
mitk::Image::Pointer mitk::ImageGenerator::MakeImage()
{
  // already in namespace mitk here!
  Image::Pointer image = Image::New();
  ImageDecorator::Pointer decorator = ImageDecorator::New();
  decorator ->Decorate( image );
  return image;
}
```

Code Layout and Indentation

- Each line of code should take no more than 120 characters.
- Use lots of whitespace to separate logical blocks of code, intermixed with Comments
- **DO NOT USE TABS.** Set up your editor to insert 2 spaces.
- Declaration of variables should be one declaration per line

```
int  counter, imageNumber;
```

```
int  counter;
int  imageNumber;
```

Class Layout

```
/*==========================================================================
Program: Medical Imaging & Interaction Toolkit
Module: $RCSfile$
Language: C++
Date: $Date: $
Version: $Revision: $
Copyright (c) German Cancer Research Center, Division of Medical and
Biological Informatics. All rights reserved.
See MITKCopyright.txt or http://www.mitk.org/copyright.html for details.
This software is distributed WITHOUT ANY WARRANTY; without even
the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the above copyright notices for more information.
==========================================================================*/
```

Copyright

```
#ifndef mitkClassName_h
#define mitkClassName_h
```

#include Guards

```
#include "... .h“
```

Includes [A…Z]

```
/*!
\brief mitkClassName
\sa .....
\verbatim
Last contributor: $Author: $
\endverbatim
*/
```

Doxygen

Class Layout

```
namespace mitk
{
template <class TType>
class ClassName : public ImageBase<VImageDimension>
{

public:
    ....typedefs....

public:
    ....methods....
protected:
    ....methods....
private:
    ....methods....

signals:
    Signal…();

public slots:
    On...();
protected slots:
    On...();

private/protected:
    ....class data members....
};
}
#endif // mitkClassName_h
```

Namespace

Class (Template)

Typedefs

Methods

QT Signals

QT Slots

Data Members

#include Guards

Use of Braces { }

- Used to delimit the scope of an if, for, while, switch.
- Braces are placed on a line by themselves

```
for ( unsigned int i = 0; i < 3; ++i )
{
    ... do something ...
}
```

```
if ( condition )
{
    ... do something ...
}
else if ( other condition )
{
    ... do something ...
}
else
{
    ... do something ...
}
```

- You can choose to use braces on a line with a code block when the block consists of a single line

```
if ( condition ) { foo = 1; }
else if ( condition2 ) { foo = 3; }
else { return; }

for (unsigned int i = 0; i < 3; ++i ) { x[i] = 0.0; }
```

#include Guards

- #inlcude guard is a particular construct used to avoid the problem of double inclusion when dealing with the #include directive.

ClassName_h

File: grandfather.h

```
struct foo
{
   int m_Member;
};
```

File: father.h

```
#include "grandfather.h"
```

File: child.h

```
#include "grandfahther.h"
#include "father.h"
```

File: grandfather.h

```
#ifndef grandfather_h
#define grandfather_h
struct foo
{
   int m_Member;
};
#endif
```

File: father.h

```
#include "grandfather.h"
```

File: child.h

```
#include "grandfahther.h"
#include "father.h"
```

KWStyle

- KWStyle is integrated in the software process to ensure that the code written by several users is consistent and can be viewed/printed as it was written by one person.

- KWStyle is primarily checking C/C++ source code and it assumes that the code is syntaxically correct, i.e it compiles on a standard compiler.

- Among the fearures provided by KWStyle:

  Several Indentation checking
  Copyright Header correctness
  Maximum line length
  Internal variable checking via regular expressions
  New line at the end of file

KWStyle

- KWStyle can be integrated with Dart as part of the dashboard (Nightly Checked Projects: ITK, VTK, Cmake, IGStk)

Filename	LEN	IVR	IVA	SEM	DCL	EOF	TAB	ESP	IND	HRD	DEF	TDR	TDA	NMS	NMC	WCM	MCM	EML	TPL	OPS
	/projects/KWStyle/Insight/Code/Common																			
itkBSplineDeformableTransform.h	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
itkChainCodePath2D.h	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
itkConceptChecking.h	0	0	0	0	0	0	0	0	57	0	0	0	6	0	0	0	0	0	8	0
itkConstShapedNeighborhoodIterator.h	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
tkCoreAtomImageToDistanceMatrixProcess.txx	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
itkCovarianceImageFunction.txx	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0
itkCovariantVector.cxx	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	3	0	0
itkCovariantVector.txx	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	3	0	0
itkDataObject.h	0	0	0	0	0	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0
itkDataObjectDecorator.h	0	0	0	0	0	1	0	0	0	0	0	0	4	0	0	0	0	0	0	0
itkDataObjectDecorator.txx	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
itkDecisionRuleBase.cxx	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
itkDecisionRuleBase.h	0	0	0	2	0	1	0	0	0	0	0	0	3	0	0	0	0	0	0	0
itkDefaultDynamicMeshTraits.h	0	0	0	0	0	0	0	0	4	0	0	0	2	0	0	0	0	0	0	0
itkDefaultImageTraits.h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
itkDefaultPixelAccessor.h	0	0	0	2	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
itkDefaultPixelAccessorFunctor.h	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
itkDefaultStaticMeshTraits.h	0	0	0	0	0	0	0	0	3	0	0	0	4	0	0	0	0	0	0	0
itkDefaultVectorPixelAccessor.h	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
itkDefaultVectorPixelAccessorFunctor.h	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
itkDenseFiniteDifferenceImageFilter.h	0	0	0	0	0	0	0	0	2	0	1	0	3	0	0	0	0	0	0	0
itkDenseFiniteDifferenceImageFilter.txx	0	0	0	0	0	0	0	1	0	0	1	0	4	0	0	0	0	0	0	0
itkDerivativeOperator.h	0	0	0	0	0	1	0	0	2	0	0	0	1	0	0	0	0	0	0	0