12/15/2010

MITK Datatypes

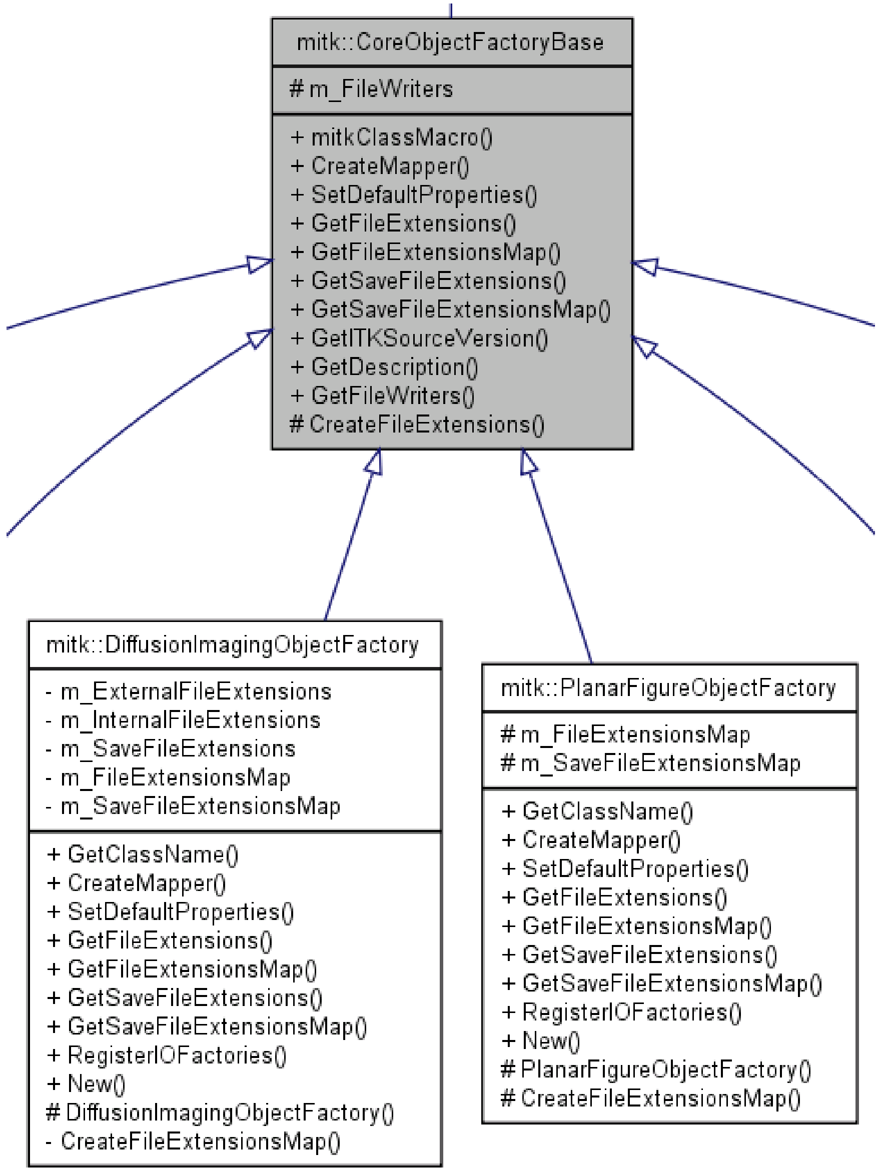
GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

Datatypes

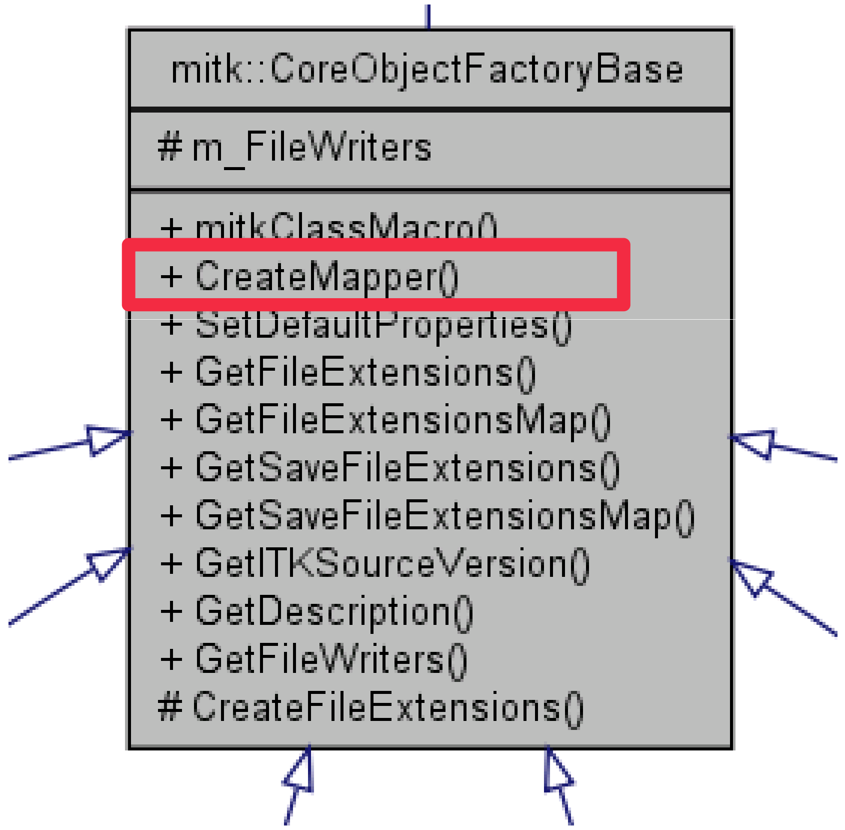
- MITK core allows modular extension
  - Datatypes
  - Readers/Writers
  - Mappers

OBJECT FACTORIES

Define an ObjectFactory

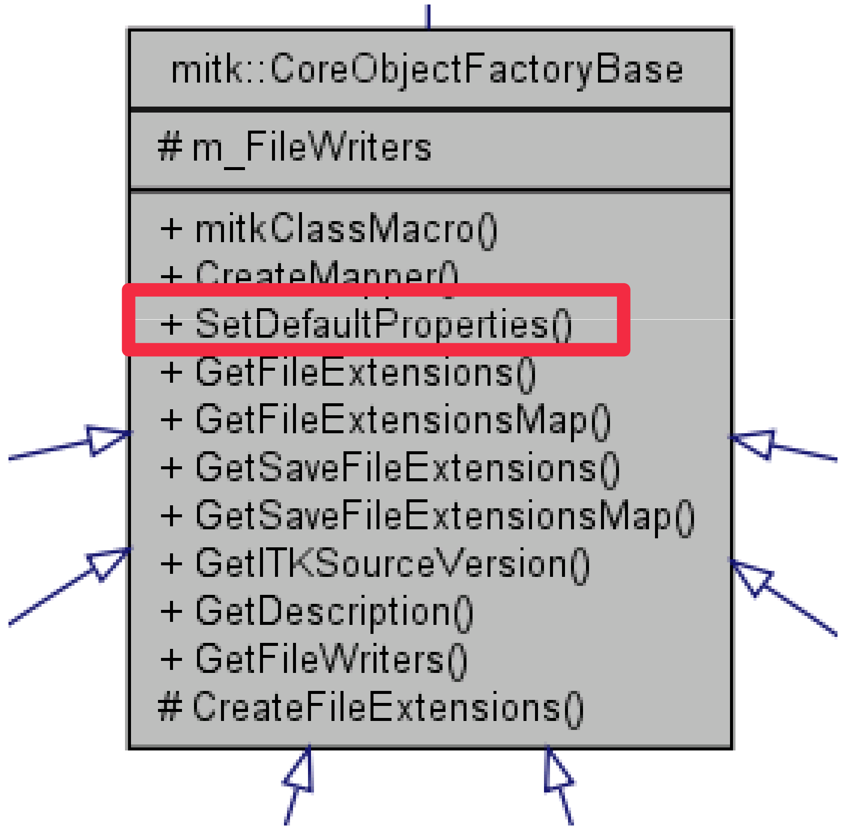


- Each Module that introduces datatypes has ONE of these
- Registers reader and writer factories for each datatype
- Hold a list of filewriters, usually one for each datatype
- Registers itself in the CoreObjectFactory as extra-factory

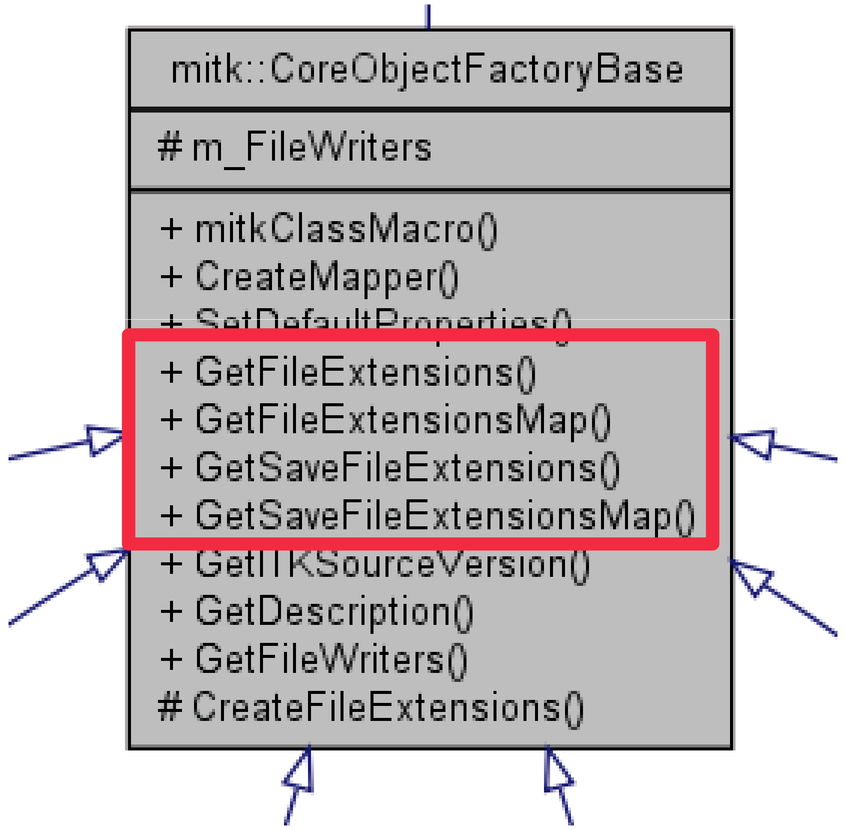


- Virtual (must override)
- Creates and returns the right mapper for a given datanode and a given Mapper-Slot-ID

```
if ( id == mitk::BaseRenderer::Standard2D )
 {
   std::string classname("QBallImage");
   if(node->GetData() &&
    classname.compare(node->GetData()-
    >GetNameOfClass())==0)
   {
     newMapper =
    mitk::CompositeMapper::New();
     newMapper->SetDataNode(node);
     node->SetMapper(3,
    ((CompositeMapper*)newMapper.GetPointer())
    ->GetImageMapper());
   }
}
```



- Virtual (must override)
- Sets properties for a given node
- Usually just passes the call to the responsible mappers who know their required properties

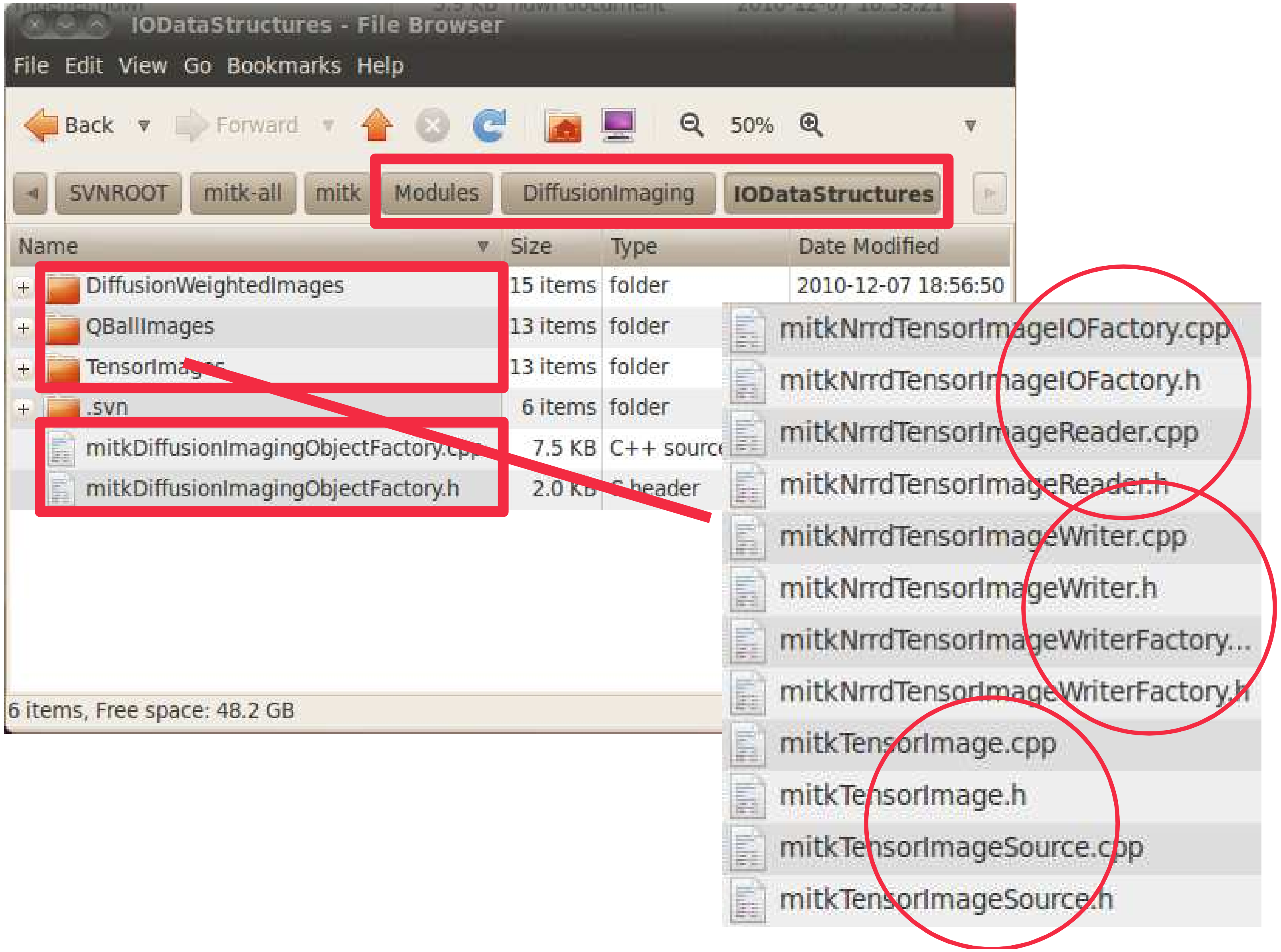


- Virtual (must override)

- Supported file extensions that correspond to the introduced datatypes

FOR EACH DATATYPE ...

Defining a new datatype



Name	Size	Type	Date Modified
DiffusionWeightedImages	15 items	folder	2010-12-07 18:56:50
QBallImages	13 items	folder	
TensorImages	13 items	folder	
.svn	6 items	folder	
mitkDiffusionImagingObjectFactory.cpp	7.5 KB	C++ source	
mitkDiffusionImagingObjectFactory.h	2.0 KB	C header	

6 items, Free space: 48.2 GB

mitkNrrdTensorImageIOFactory.cpp
mitkNrrdTensorImageIOFactory.h
mitkNrrdTensorImageReader.cpp
mitkNrrdTensorImageReader.h
mitkNrrdTensorImageWriter.cpp
mitkNrrdTensorImageWriter.h
mitkNrrdTensorImageWriterFactory...
mitkNrrdTensorImageWriterFactory.h
mitkTensorImage.cpp
mitkTensorImage.h
mitkTensorImageSource.cpp
mitkTensorImageSource.h

Inheriting from itk::ObjectFactoryBase

```
class MitkDiffusionImaging_EXPORT NrrdDiffusionImageIOFactory : public itk::ObjectFactoryBase {
public:
  /** Not shown: Standard class typedefs. */
  /** Class methods used to interface with the registered factories. */
  virtual const char* GetITKSourceVersion(void) const;
  virtual const char* GetDescription(void) const;
  /** class instantiation and type information. */
  itkFactorylessNewMacro(Self);
  static NrrdDiffusionImageIOFactory* FactoryNew() { return new NrrdDiffusionImageIOFactory;}
  itkTypeMacro(NrrdDiffusionImageIOFactory, ObjectFactoryBase);

  /** Register one factory of this type  */
  static void RegisterOneFactory(void)  {
    static bool IsRegistered = false;
    if ( !IsRegistered )    {
      NrrdQBallImageIOFactory::Pointer fac = NrrdQBallImageIOFactory::New();
      ObjectFactoryBase::RegisterFactory( fac );
      IsRegistered = true;
    }
  }
  /** not shown: protected constructor/destructor */
};
```

Constructor of reader factories

```
NrrdTensorImageIOFactory::NrrdTensorImageIOFactory()
{
  this->RegisterOverride(
    "mitkIOAdapter",
    "mitkNrrdTensorImageReader",
    „TensorImages IO",
    1,
     itk::CreateObjectFunction<IOAdapter< NrrdTensorImageReader> >::New()
  );
}
```

Constructor of writer factories

```
NrrdTensorImageWriterFactory::NrrdTensorImageWriterFactory()
{
  this->RegisterOverride(
    "IOWriter",
    "NrrdTensorImageWriter",
    "NrrdTensorImage Writer",
    1,
    itk::CreateObjectFunction< mitk::NrrdTensorImageWriter >::New()
  );
}
```

FRAMEWORK VIEW OF THINGS

CommonFunctionality::SaveBaseData()

- How the framework accesses the registered writers?

```
mitk::CoreObjectFactory::FileWriterList fileWriters =
  mitk::CoreObjectFactory::GetInstance()->GetFileWriters();

bool writerFound = false;

for (mitk::CoreObjectFactory::FileWriterList::iterator it =
  fileWriters.begin() ;
         it != fileWriters.end() ; ++it)    {
           if ( (*it)->CanWriteDataType(data) ) {
             writerFound = true;
             SaveToFileWriter(*it, data, NULL, aFileName);
             /* … */
           }
         }
```

BaseDataIO::LoadBaseDataFromFile()

- How the framework accesses the registered readers?

```
std::list<LightObject::Pointer> allobjects =
   itk::ObjectFactoryBase::CreateAllInstance("mitkIOAdapter");

for( std::list<IOAdapterBase::Pointer>::iterator k = possibleIOAdapter.begin();
                                              k != possibleIOAdapter.end();  ++k )   {
   if((*k)->CanReadFile(path, filePrefix, filePattern) )
   {
     BaseProcess::Pointer ioObject =
         (*k)->CreateIOProcessObject(path, prefix, pattern);
     ioObject->Update();
     BaseData::Pointer baseData =
         dynamic_cast<BaseData*>(ioObject->GetOutputs()[0].GetPointer());
   }
}
```

Other Object Factories

- What about the following factories?
  - **QMCore**ObjectFactory
  - **SBCore**ObjectFactory
  - **CoreExt**ObjectFactory
- Same as previous example:
  - They Register all datatypes belonging to QM, SB, or extended Core
  - Workaround to include the old factory hierarchy in the new modular system
  - Should be replaced by smaller object factories that register datatypes for different topics

MODULE ACTIVATION

Module Activation Problem

- Activation of modules so far only possible by Bundles that actually require the module
- To register the datatypes early enough, we need an activator bundle that starts together with the framework

```
Manifest-Version: 1.0
Bundle-Name: MBI DiffusionImaging
Bundle-SymbolicName: org.mitk.diffusionimaging
…
Bundle-Activator: mitk::DiffusionImagingActivator
Bundle-ActivationPolicy: eager
```

Bundle Start-Method

```
void mitk::DiffusionImagingActivator::Start(berry::IBundleContext::Pointer  /*ctx*/)
{
  RegisterDiffusionImagingObjectFactory();

  QmitkNodeDescriptorManager* manager =
    QmitkNodeDescriptorManager::GetInstance();

  QmitkNodeDescriptor* desc = new QmitkNodeDescriptor (
    QObject::tr("DiffusionImage"),
    QString(":/QmitkDiffusionImaging/QBallData24.png"),
    mitk::NodePredicateDataType::New("DiffusionImage"),
    manager );

  manager->AddDescriptor(desc);

  // not shown: other data types
}
```

Module activation

```
// directly written in the cpp-file of the ObjectFactory:

void RegisterDiffusionImagingObjectFactory()
{
  static bool oneDiffusionImagingObjectFactoryRegistered = false;
  if ( ! oneDiffusionImagingObjectFactoryRegistered ) {
    MITK_INFO << "Registering DiffusionImagingObjectFactory..." << std::endl;
    mitk::CoreObjectFactory::GetInstance()
      ->RegisterExtraFactory(mitk::DiffusionImagingObjectFactory::New());
    oneDiffusionImagingObjectFactoryRegistered = true;
  }
}
```

THANK YOU