3/5/2015

Introduction to the new IO System

GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

50 Years – Research for
A Life Without Cancer

Today‘s session is gonna take a little longer

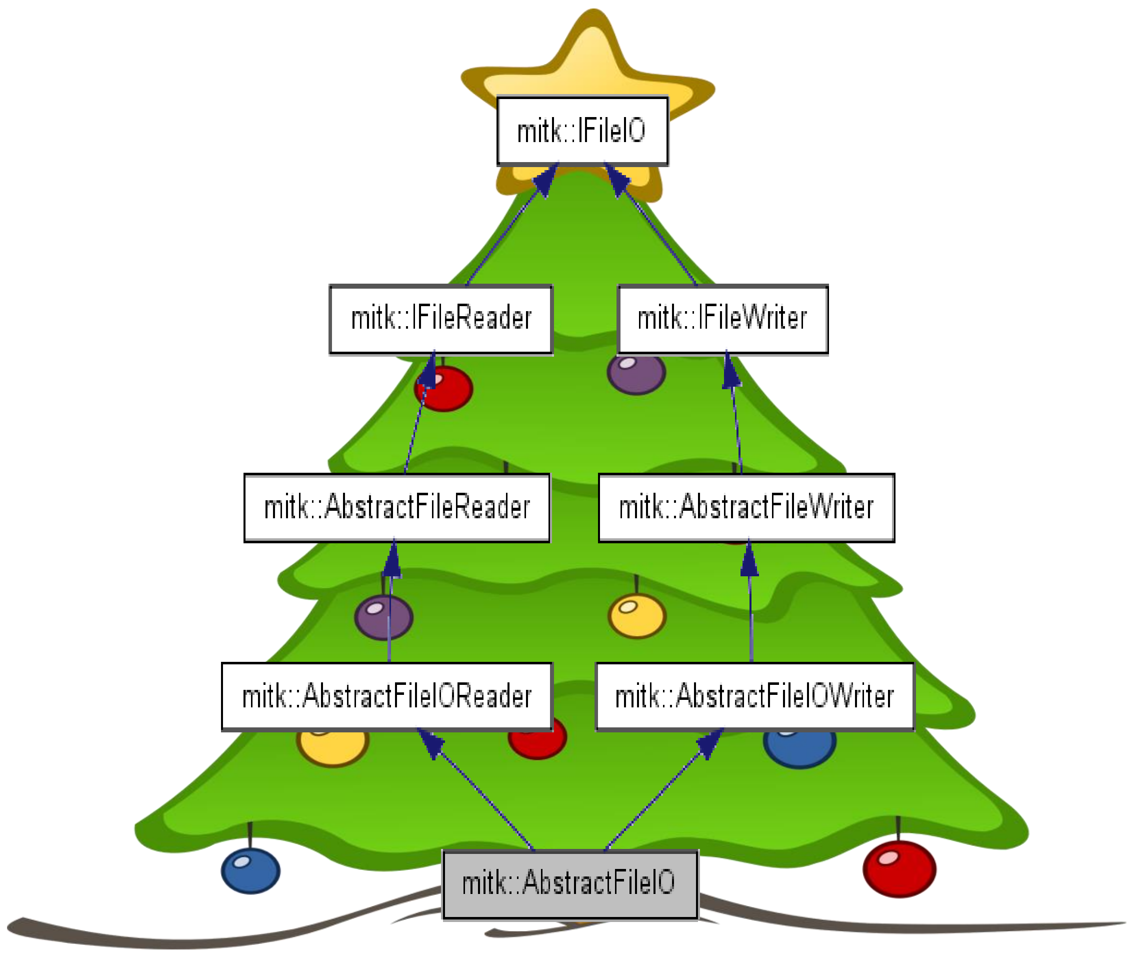
1. Features
2. The IO Architecture
3. Quick HowTo

Features

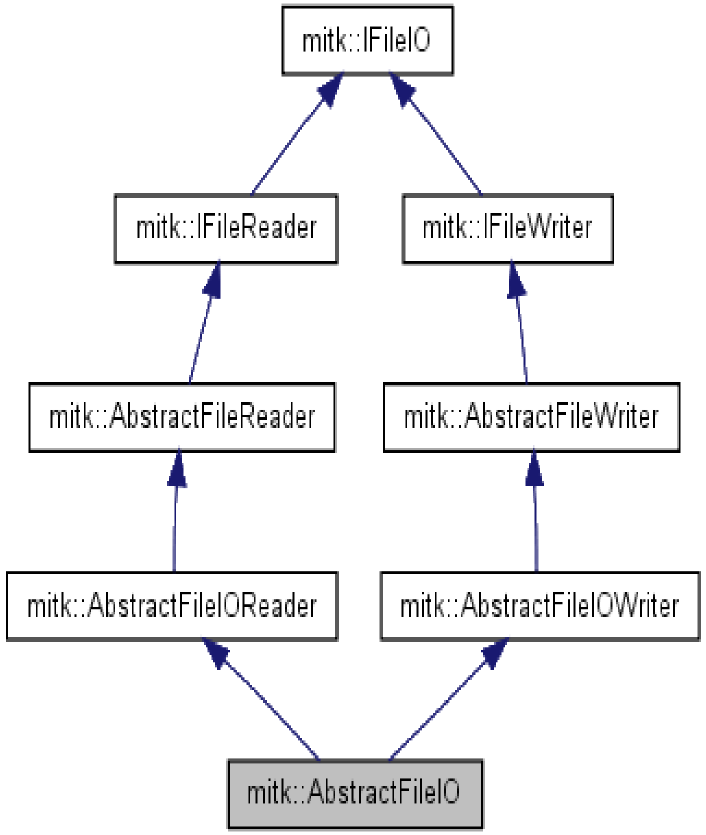
1. Common Interface for all Reader/Writer
2. Reader/Writer implementation in one file….
3. …and one method
4. User-defineable options while reading/writing files
5. Automatic handling of file location and streaming
6. Core needs no fiddeling
7. Reader/Writer can be superceded externally
8. Confidence level
9. Usage of MIME-Types
10. Fetches coffee if asked nicely

It’s like Christmas all over again!

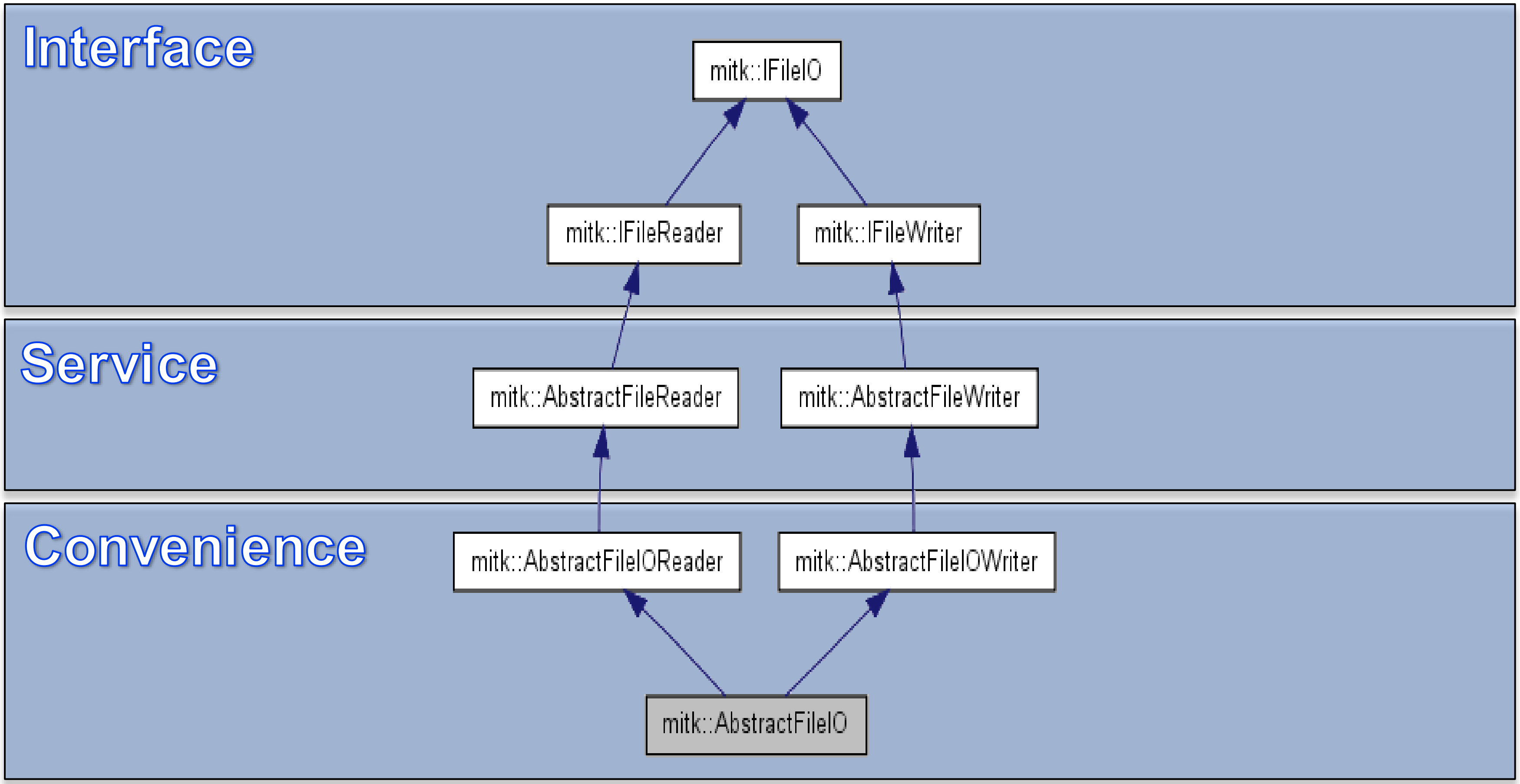
The IO Architecture



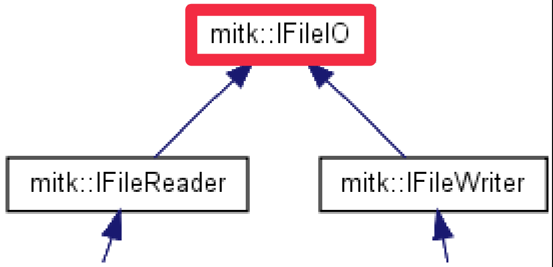
The IO Architecture



The IO Architecture



IFileIO



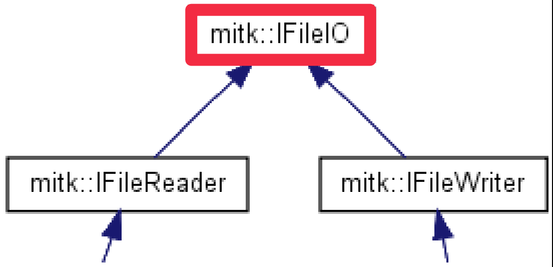
General concepts shared between Reader and Writer

1) The Confidence Level

2) Options

3) Progress Callbacks (Future Work)

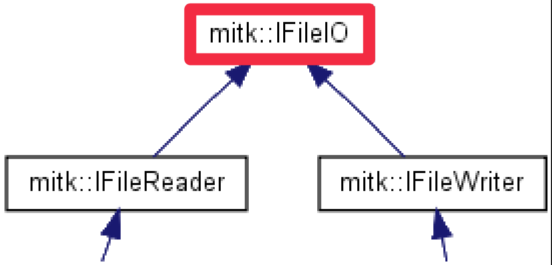
IFileIO:Confidence Level



Concept to rank Reader and Writer

```
enum ConfidenceLevel
 {
   Unsupported = 0,
   PartiallySupported = 8,
   Supported = 16
 };
```

- Your reader should return one of these values when presented with a file
- Higher confidence is preferred by the system



IFileIO:Options

Concept to control Reader/Writer behaviour

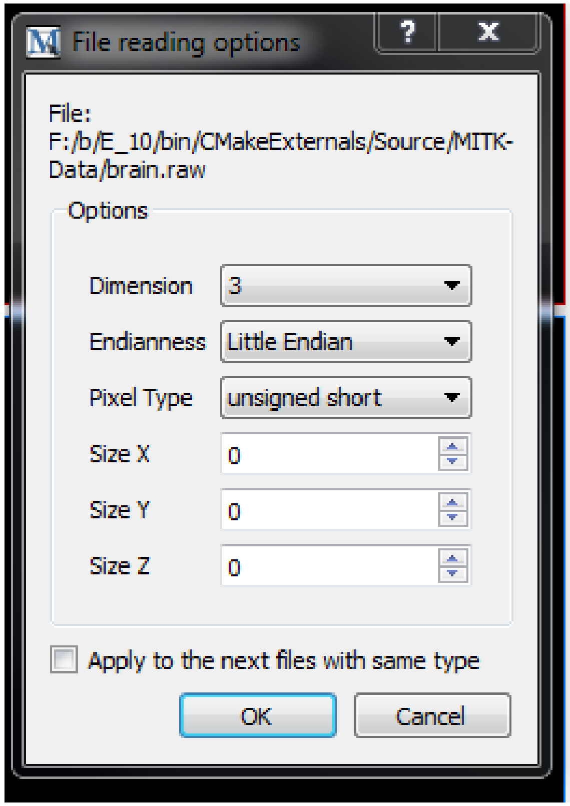
```
typedef std::map<std::string, us::Any> Options;

virtual Options GetOptions();
virtual void SetOptions(const Options& options);
virtual us::Any GetOption(const std::string& name);
virtual void SetOptions(const Options& options);
```

- Reader/Writer should define their default options
- Example Implementation: RawImageFileReaderService

IFileIO:Options

Options are

Converted into a form when opening a file via a GUI!



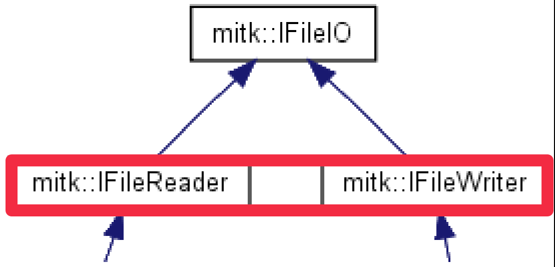
File:
F:/b/E_10/bin/CMakeExternals/Source/MITK-Data/brain.raw

Options

Dimension	3
Endianness	Little Endian
Pixel Type	unsigned short
Size X	0
Size Y	0
Size Z	0

☐ Apply to the next files with same type

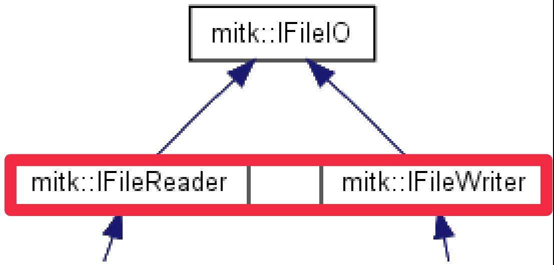
IFileReader & IFileWriter



General concepts specific to Reader and Writer

1) Defining locations and abstracting from location to streams

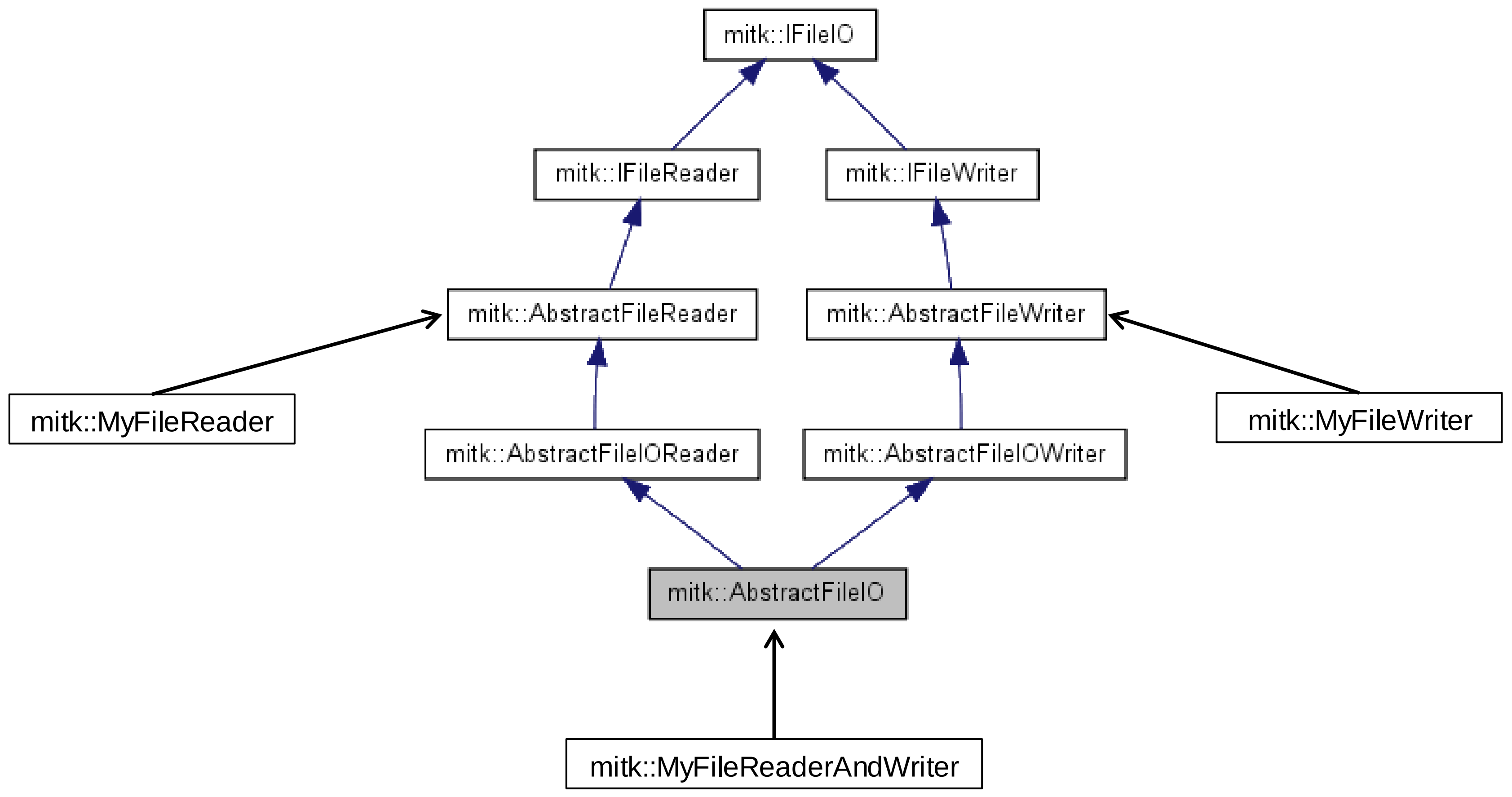
2) Read / Write methods



Locations and Streams

- Reader/Writer should be indifferent towards Stream/FilePath
- Interfaces require Reader/Writer to handle both!

Implement own IO: Use Abstract Classes



Benefits: Deriving from Abstract Classes

- Handels Stream / Filepath abstraction
- Registration in Reader/Writer registry system
- Sensible default implementations
- Avoid code duplication
- Fast and easy IO implementation

Reader/Writer Registry

- Abstract Classes implemented as Microservice
- Reader/Writer globally available
  - Available from GUI
  - Easy-Peasy file reading from code via IOUtil

```
std::vector<mitk::BaseData::Pointer> result;
result = IOUtil::Load("/Path/To/My/Unicorn");
```

Reader/Writer Registry

- Supercession of Readers
  - Use Confidence and Priority to select best reader

- Reader/Writer globally available
  - Available from GUI
  - Easy-Peasy file reading from code via IOUtil

```
std::vector<mitk::BaseData::Pointer> result;
result = IOUtil::Load("/Path/To/My/Unicorn");
```

Further Reading

- For more detailed info, please consult the concept page!

Thank you for your attention!

Further information on www.dkfz.de

GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

50 Years – Research for
A Life Without Cancer