A Simple Command Line Argument Parser

Release 1.0

Marius Staring[1] and Stefan Klein[2]

April 13, 2011

[1]Division of Image Processing, Leiden University Medical Center, Leiden, The Netherlands
[2]Biomedical Imaging Group Rotterdam, Departments of Radiology & Medical Informatics, Erasmus MC, Rotterdam, The Netherlands

Abstract

This document describes the implementation of a simple command line argument parser using the Insight Toolkit ITK www.itk.org. Such a parser may be useful for use in the examples of the ITK.

This paper is accompanied with the source code.

Latest version available at the Insight Journal [http://hdl.handle.net/1926/3258]

Contents

1 Introduction

Command line argument parsing is a common task for many (small) programs. There are many tools already available for parsing command line options. These tools, however, typically require the inclusion of a (large) library (Boost, Wx, etc). The advantage of our proposed parser is that it is very small and non-intrusive while remaining versatile enough to fulfill the needs of the ITK examples. The proposed parser is used extensively for many years now in the toolkit `praxix`: http://code.google.com/p/praxix/. It is extremely simple, with only three functions. One to set the command line arguments, and two to get them. Arguments are set with:

```
parser->SetCommandLineArguments( argc, argv );
```

Arguments are fetched using

```
parser->GetCommandLineArgument( "-key", argument );
```

You can also just check to see if an argument has been passed without retrieving its value with

```
parser->ArgumentExists( "-key" );
```

The tool assumes that arguments are passed in key-value combinations, for example `-key value1` $\cdots$ `valueN`. Keys are identified as a "-" followed by a string; subsequent entries that are not keys are the values. Zero or more values can be specified. The functions `GetCommandLineArgument` and `ArgumentExists` return a boolean: true when `-key` was provided and false otherwise.

1.1 Default values

Arguments can be initialized to default values, which will be left untouched if the key is not provided at the command line. If an argument is initialized with a vector of `size` > 1, and if only one (1) argument is provided in the command line, a vector of size `size` is created and filled with the single argument.

1.2 Internals

Internally, the command line arguments are stored in an `std::map` of the argument or key as an `std::string` together with the index. We make use of the casting functionality of string streams to automatically cast the stored string to the requested type.

2 Examples of usage

In this section we demonstrate the ease with which the parser can be constructed and used for many different types of arguments.

2.1 Constructing the parser

The command line argument parser is constructed as follows.

```
#include "itkCommandLineArgumentParser.h"
 ...
// Create a command line argument parser
itk::CommandLineArgumentParser::Pointer parser = itk::CommandLineArgumentParser::New();
parser->SetCommandLineArguments( argc, argv );
```

2.2 Passing a boolean argument

A boolean argument is set to true by simply passing the flag:

```
executablename -z
```

This code

```
bool compress = parser->ArgumentExists( "-z" );
```

will return true if `-z` was part of the list of arguments, and false otherwise.

2.3 Passing an integer argument

```
executablename -num 5
```

```
int intValue; // no default
bool retnum = parser->GetCommandLineArgument( "-num", intValue );
```

2.4 Passing a floating argument

```
executablename -pi 3.1415926
```

```
float pi = 3.0; // default
bool retpi = parser->GetCommandLineArgument( "-pi", pi );
```

2.5 Passing a string argument

```
executablename -m Wavelet
```

```
std::string method = "Fourier"; // default
bool retm = parser->GetCommandLineArgument( "-m", method );
```

2.6 Passing a vector argument

```
executablename -in file1.png file2.bmp
```

```
std::vector<std::string> inputFileNames; // no default
bool retin = parser->GetCommandLineArgument( "-in", inputFileNames );
```

2.7 Setting argument defaults

Behavior when the argument is passed:

```
executablename -pA 2 5 9 4
```

```
std::vector<int> vecA( 3, 1 ); // using default values
bool retpA = parser->GetCommandLineArgument( "-pA", vecA );
// The value of vecA is {2, 5, 9, 4}
```

Behavior when the argument is NOT passed:

```
executablename
```

```
std::vector<int> vecA( 3, 1 ); // using default values
bool retpA = parser->GetCommandLineArgument( "-pA", vecA );
// The value of vecA is {1, 1, 1}
```

2.8 Combined example

Multiple arguments can be passed to the command as follows:

```
executablename -in input1.mhd input2.mhd -z -out output.png -pi 3.1415926535
```

These arguments are retrieved identically as in the individual examples above.

3 Conclusion

This document describes the implementation of a simple command line argument parser using the Insight Toolkit ITK www.itk.org. Such a parser may be useful for use in the examples of the ITK or on it's wiki.