

MetaCommand Documentation

Authors: Julien PEYRE and Julien JOMIER

Institution: Computer-Aided Diagnosis and Display Lab @ UNC Chapel Hill

Last Updated: September 15, 2005

1) Introduction:

Thanks to MetaCommand you can easily receive parameters in your software from the command line. The purpose of this class is to provide a unified framework for command line parsing.

For instance, all programs that use MetaCommand are self-described using the command line `-vxml`:

```
$ ./AtlasSummation -vxml
<option>
<number>0</number>
<name>Outputcount</name>
<tag>oc</tag>
<description>Image of count values for each voxel</description>
<required>0</required>
<nvalues>1</nvalues>
<field>
<name>filename</name>
<description></description>
<type>string</type>
<value></value>
<external>0</external>
<required>1</required>
...
```

Moreover, The description of the parameters can be easily retrieve at runtime via `./your_program -v` :

```
$ ./AtlasSummation -v
Usage : d:\Work\itkUNCApplications-VC++\bin\debug\AtlasSummation.exe
[-oc <filename> : Image of count values for each voxel]
[-ao : Adjust the mean origin to the input images]
[-as : Adjust the mean size to fit the input images]
[-stdev <bool> (true) : Is the sigma image standard deviation?]
[-LT [number] (4) : lowerThreshold : Minimum number of contributing images for pixel
to be counted in output]
[-OS <X> <Y> <Z> : outputImageSize X Y Z : output size (can't use with adjust size)]
```

[-OSp <X> <Y> <Z> : output spacing (default- initial image spacing)]
<infile> : infile filename
<outputmean> : output mean filename
<outputvar> : output variance filename

All these advantages are provided for you in the MetaCommand class.

2) MetaCommand in use:

a) Definitions:

Metacommand differentiates the term 'Option' and 'Field'. Basically an option can be placed anywhere in the command line as long as it is defined by a tag. On the other hand, the fields are placed in order and do not require any tag.

b) Programmer' point of view:

To use MetaCommand you have to include the file *metaCommand.h* in your code.

```
#include <metaCommand.h>
```

Then you need to declare a MetaCommand object, for example:

```
MetaCommand command;
```

To add an option (a tag + a field) you have to use 2 methods:

- To define an option:
 - `SetOption(std::string name, std::string tag, bool required, std::string description)`, there are two more arguments (the type = flag, and the default value="") but it is not necessary to put them because they are initialized correctly by default. The first argument is the name of the tag, the second is the string to write in your command line to use this option, the third indicates if the tag is required or not, and the last one is a description of the tag.

- `AddOptionField(std::string optionName, std::string Name, MetaCommand::TypeEnumType Type, bool required, std::string defValue)`.

The first argument indicates at which tag the field belongs giving the name of the tag (see first argument of `SetOption`), the second argument is the name of the field, the third one is the type of the field (bool, int, float or string), the fourth argument indicates if the field is required or not, and the last one is the default value (note that you have to put a string even if it is your type is int or float).

NOTE: you can use several times the method `AddOptionField` for a same tag.

- To define a field:

You can also just add a field (without the tag) thanks to the following method:
`AddField(std::string Name, std::string Description, MetaCommand::TypeEnumType Type, bool ExternalData)`.

The first argument is the name of the field, the second his description, the third one his type and the last one indicates if that data is external to your application or not (e.g : a filename is external but the size of the input image is internal).

c) Example:

Let's define a command line like: `./example input.mha -w 2 output.mha`

This will be coded as:

```
MetaCommand command;
```

```
command.SetOption("Write", "w", false, "writes the current image to the designated file with a type");  
command.AddOptionField("Write", "filename", MetaCommand::STRING, true);  
command.AddOptionField("Write", "Type", MetaCommand::INT, false, "1"); //by default type=1  
command.AddField("infile", "infile filename", MetaCommand::STRING, true);
```

Then the user can parse the command line thanks to the method:

```
Parse(int argc, char* argv[])
```

```
if( !command.Parse(argc, argv) )  
{  
    return 1;  
}
```

Finally to access the different options, one of these 4 methods can be used:

- `GetValueAsString(Option option, std::string fieldName)`
- `GetValueAsFloat(Option option, std::string fieldName)`
- `GetValueAsInt(Option option, std::string fieldName)`
- `GetValueAsBool(Option option, std::string fieldName)`

To recover the parameters of an option (tag + field), in the first argument of these methods the name of the tag has to be specified and in the second argument the name of the field.

To recover the parameter of a single field you just need to specify the first argument: the name of the field.

Example: to access the parameters of the previous example :

```
std::string imageIn = command.GetValueAsString("infile");  
std::string imageOut = command.GetValueAsString("Write", "filename");  
int outputType = command.GetValueAsInt("Write", "Type");
```