

Using the MiPal Whiteboard Classgenerator

Mick Hawkins

14 October 2015

Contents

1 Introduction	2
1.1 Supported Operating Systems	2
2 Creating an input file	2
2.1 File type and filename	2
2.2 Specifying your name	2
2.3 Specifying an alias	3
2.4 Specifying simple variables	3
2.4.1 Specifying arrays	3
2.5 Specifying the struct comment	4
3 Installing the classgenerator executable file	4
4 Running the program	4
5 Supported data types	5

1 Introduction

The Classgenerator is a command line tool used to generate classes for use with the MiPal Whiteboard. It reads input from a text file and generates Whiteboard .c and .h class files, and a C++ wrapper.

It is assumed the user has general skills in use of the `bash` shell.

1.1 Supported Operating Systems

The Classgenerator requires MacOS X 10.9 and later.

2 Creating an input file

An input file must be created before using the Classgenerator. The input file specifies the variables types used in the generated classes.

2.1 File type and filename

The input file must be a plain-text `.txt` file. The `.txt` file extension must be used.

To correctly generate C and C++ class names:

- The filename should use lowercase letters
- The filename must begin with a lowercase letter
- The filename should use underscores between words
- Numbers may be used
- Other than in the `.txt` file extension, periods/fullstops must not be used

If the input file's name includes uppercase letters:

- These capital letters will be kept and used in the C++ filename/class (which is in camel case)
- These capital letters will be converted to lowercase for the `wb_` files

These are some examples of *suitable* filenames:

```
ball_colour.txt
oculus_prime_interface.txt
vision_goals.txt
point2D.txt
point_2D.txt
```

These are examples of *unsuitable* filenames:

```
BallColour.txt
goal.doc
WALK.txt
vision_goals
```

A sample text file `MY_test.txt` can be found in the `GUNao/posix/classgenerator/classgenerator` folder.

2.2 Specifying your name

As the author you may, as an option, specify your name in the input file. Your name is used in the comment at the top of each file:

- As the creator of the file
- In the copyright clause
- In the GNU license

If you not specify your name in the input file, the system username will be used.

Specify your name in the first line of the input file using the following format:

```
author /tab Your Name
```

- author must be in lowercase
- There must be a single tab between author and your name
- Your name must be written exactly how you want it to appear (as a suggestion, capitalised with a space between parts of the name)

Hyphenated names, and multi-word names will work as expected.

Examples of how to specify names:

```
author Captain Spaulding
```

```
author Otis B. Driftwood
```

2.3 Specifying an alias

To allow compatibility with existing, older code, you may specify an alias class using the following format:

```
alias /tab alias_filename
```

2.4 Specifying simple variables

To specify variables, use the following format:

```
datatype /tab variable_name=default /tab comment
```

- Each variable must include a datatype, variable name and comment
- The data type must be written as specified in [section 5 Supported Data Types](#)
- There must be a single tab between each of the datatype, variable name/default, and comment
- Variable names should be written exactly how you want them to appear
- Specifying a default value is optional (see below*)
- The comment may be specified by “//” to aide in readability

Currently supported data types are listed in [section 5 Supported Data Types](#). Strings and objects to be added shortly.

Example of specifying variables in an input file:

```
int16_t    pointX=5           // pointX is the X coordinate
int16_t    pointY            // pointY is the Y coordinate
bool       is_red=false      is_red is true if the colour is red
```

In this example, pointY does not have a default value specified

Note: depending on the tab setting of your text file editor, things may not line up perfectly.

*If default values are not specified, the following defaults will be used:

- Boolean: false
- Numerical types: 0

2.4.1 Specifying arrays

To specify arrays, use the following format:

```
datatype /tab variable_name[array_size]=default /tab comment
```

The specifications in 2.4 also apply to arrays. In addition:

- Default values for arrays may be specified between curly braces and separated by commas
- If default values are not specified, the array will be filled with the default value of the variable type (as specified in 2.4)
- An array size must be specified

- If a string is specified as an array of `char`, the array size must allow for the null termination character

Example of specifying arrays in an input file:

```
int16_t    my_values[4]={1,1,2,3}    // my_values has default values
int16_t    your_values[3]            // no default specified
```

In this example, `your_values` does not have a default value specified

2.5 Specifying the struct comment

To specify a comment for the struct, leave a blank line after the variables, and enter the comment lines at the end of the text file. An example of an input text file with a struct comment is:

```
int16_t    pointX=5                // pointX is the X coordinate
int16_t    pointY                  // pointY is the Y coordinate
/return <- this is a blank line
```

This is the first line of a comment for the struct.

This is the second line.

This comment will appear above the struct in the `wb_header` file and the C++ wrapper

3 Installing the classgenerator executable file

The classgenerator executable is located in the `GUNao/posix/classgenerator/classgenerator` folder. It is called `classgenerator`.

To allow the executable to be run from any directory, copy it to the `usr/local/bin` directory under MacintoshHD. This directory is hidden. To open it, go to the Finder and, under the “Go” menu, use “Go to folder”.

If you do not have a `usr/local/bin` directory, enter the following in the Terminal:

```
sudo mkdir -p /usr/local/bin
cd /usr/local/bin
open .
```

...this will create and open the directory. Copy the executable into this folder.

4 Running the program

With the program installed in the `usr/local/bin` directory, it can be run from any location.

In the Terminal, change to the directory that you would like your generated files to be located. Put your input file in this directory also.

The name of the input file must be entered as a command line argument. For example:

```
classgenerator ball_colour.txt
```

This will run the generator using the file `ball_colour.txt` as input and will generate the Whiteboard classes:

```
wb_ball_colour.h
wb_ball_colour.c
```

To also generate a C++ wrapper for these files, use the command line argument `c` or `-c`

```
classgenerator ball_colour.txt c
```

This will generate the Whiteboard classes and a C++ wrapper:

```
wb_ball_colour.h  
wb_ball_colour.c  
BallColour.h
```

The command line arguments may be entered in any order. These variations will produce the same result:

```
classgenerator ball_colour.txt -c  
classgenerator c ball_colour.txt  
classgenerator -c ball_colour.txt c
```

Note: Command line options for a Swift wrapper and usage information will be added shortly.

5 Supported data types

Strings, arrays and object types to be supported shortly. The currently supported data types are:

```
bool  
  
char  
signed char  
unsigned char  
  
int  
signed int  
unsigned  
unsigned int  
  
int8_t  
uint8_t  
int16_t  
uint16_t  
int32_t  
uint32_t  
int64_t  
uint64_t  
  
short  
short int  
signed short  
signed short int  
unsigned short  
unsigned short int  
  
long  
long int  
signed long  
signed long int  
unsigned long  
unsigned long int  
  
long long  
long long int  
signed long long  
signed long long int  
unsigned long long  
unsigned long long int
```

long64_t

float
float_t

double
double_t

long double
double double

DRAFT