

# "set \_code.pl" v1.2 documentation for auto-completion in Flowcode's Custom \_Code.c

nils

March 1, 2008

## 1 Introduction

While there are many functions already available as components in Flowcode, sometimes one needs to implement these in a different way or wants to build completely new functions. Therefore MatrixMultimedia chose to give the user the opportunity to do so with a file called `Custom_Code.c`. If one puts into this file its C code, the new functions can easily be accessed in Flowcode via a *Component Macro*.

Sometimes this is not so easy because there are some *Macro* definitions one has to make before the function will even be accessible inside of Flowcode.

## 2 The Macro definitions in Custom \_Code.c

The following *Macro* definitions are used inside of `Custom_Code.c`:

**MacroNames** gives the functions' names and the total count of them

```
Count=<function count>
1=<1st function's name>
2=<2nd function's name>
...
```

**MacroReturns** gives the functions' return types that can be short,void, char or char\*

```
1=<1st function's type>
2=<2nd function's type>
...
```

**MacrolsPrivate** sets the individual function's flag if it is shown in the list of the *Component Macro* or not

```
1=<1st function's private flag>
2=<2nd function's private flag>
...
```

**MacroParameters\_<function's name>** for every function there is this block for the parameters i.e the function's variables

```
Count=<variable count>
1=<1st variable's name>
2=<2nd variable's name>
...
```

**MacroParamTypes\_<function's name>** this block directly follows the variables' names block and gives the type for every variable

```
1=<1st variable's type>
2=<2nd variable's type>
...
```

Later on inside of the functions are two other *Macro* implementations used. The "Start" follows the function's definition as soon as possible after the opening { and the "End" comes just before the closing }. This is necessary for Flowcode to import the C functions correctly into the project's C file.

```
/*Macro_<function>_Start*/
```

```
/*Macro_<function>_End*/
```

In order to get the selfdefined functions to work, every single one of these definitions has to be edited and spelled correctly. If not, one runs into compiling problems, Flowcode error messages, or worse, can't even access the functions via the *Component Macro*.

For this reason the Perl script `set_code.pl` was written. It collects the necessary information and sets the Macro definitions accordingly right.

### 3 The script's "work"

The abovementioned *Macro* names have to be read in from the functions and set if one will supply these to the Flowcode *Custom Component*.

The Perl script, `set_code.pl`, does this specific job if it is dropped in the same folder as `Custom_Code.c` and then being invoked on the command line or something similar.

The flowchart in Figure 1 shows the basic steps of this script:

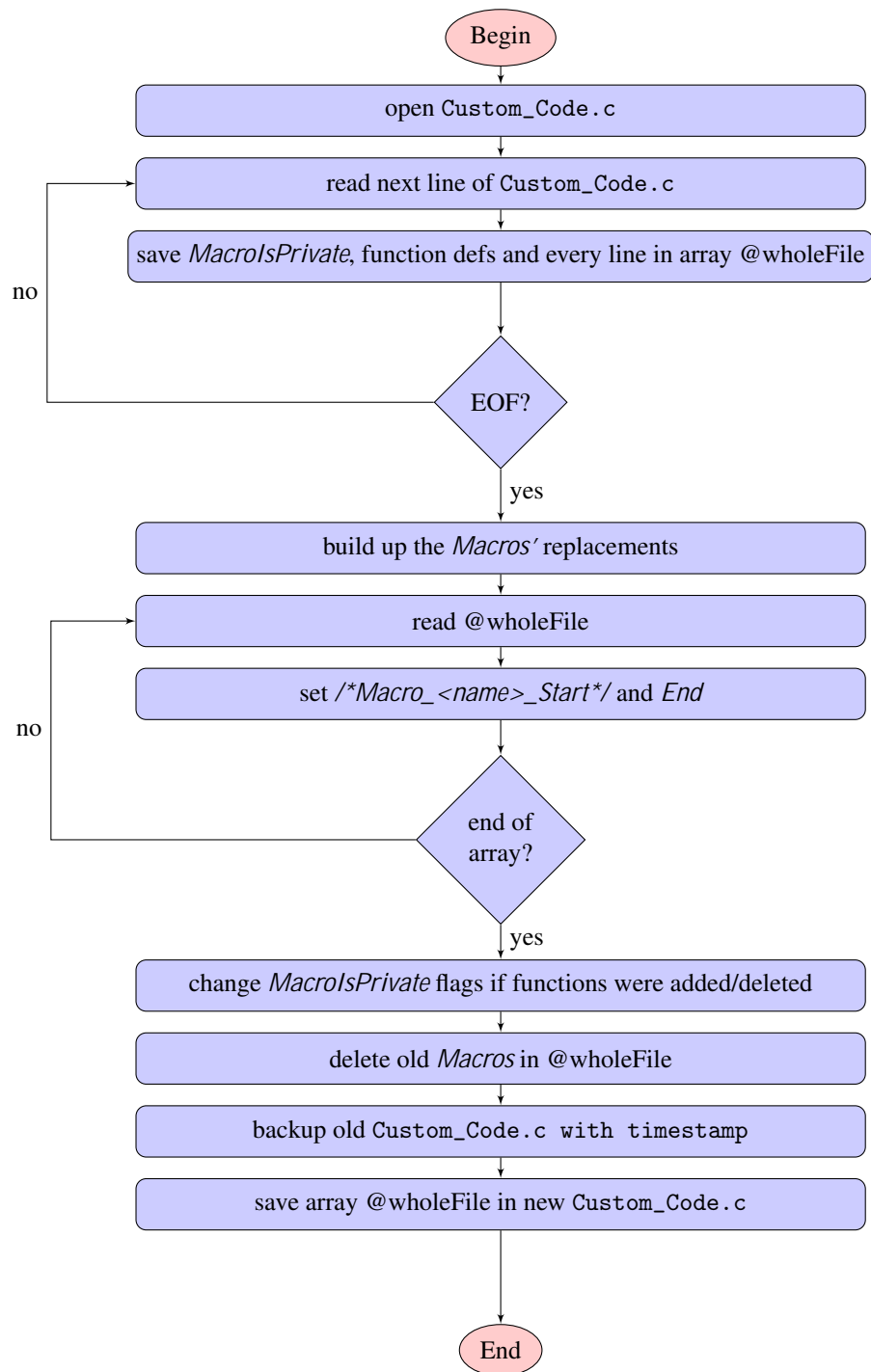


Figure 1: basic flowchart of `set_code.pl`

## 4 Usage

Even if already described in the header of `set_code.pl`, but for a complete guide now added in here as well.

If one wants to use the script, Perl is needed and available for free from <http://www.activestate.com> for all platforms. The next step is copying the script in the subdirectory `Components` of the Flowcode installation.

### 4.1 Windows

In Windows, one needs to open a command line window and change to the directory, or type in the location of the script like following examples.

```
C:\Programs\Matrix Multimedia\Flowcode\Components>perl set_code.pl ↵
```

or just type in

```
perl C:\Programs\Matrix Multimedia\Flowcode\Components\set_code.pl ↵
```

If Windows does not recognize the perl command, then it may be necessary to edit the Path environment variable by right clicking on *My Computer* and select *Settings*. Then to *Advanced -> Environment Variables -> System Variables* and add the path to the perl binary in the *Path* variable..

### 4.2 Linux/Unix

This script originated/is being developed on a BSD based system. So if the `$PATH` is set right, then go to the directory where the script is, and change the permission vector to executable , or invoke Perl with the script as parameter.

```
$ chmod u+x set_code.pl ↵
```

```
$ ./set_code.pl ↵
```

or

```
$ perl set_code.pl ↵
```

## 5 Conclusion

The script does the work of finding, generating and replacing *Macro* definitions inside the `Custom_Code.c` file and makes an automatic backup with a timestamp. It is a valuable asset if one wants to add numerous functions on a bigger project and does not have the time to edit the file when changing only minor aspects.

The size of a project's `Custom_Code.s` file doesn't matter to the script, but to the project's C file, because every function even if not used, is imported. So one should not try to make an overall usable file that is just eating up valuable memory of the microcontroller. Instead there should be a `Custom_Code.c` file for every project.

## 5.1 v1.2

Introduced with this version is the capability of handling all different kinds of brace flavours there might be. But be aware, that not everything that *is* possible, *has* to be done. So stick with a decent style. My favourite is no.1

```
void Test(short foo){
    some code;
    more code;
}
```

```
void Test(short foo)
{
    some code;
    more code;
}
```

```
void Test(short foo)
{    some code;
    more code;
}
```

```
void Test(short foo)
{    some code;
    more code;}
```

```
void Test(short foo){some code; more code;}
```

## 5.2 v1.0

At the time of writing the script is not fully compatible to everyone's style of writing functions. It should be of the following form

```
void Test(short foo){
    some code;
    more code;
}
```

to guarantee compatibility with `set_code.pl`.