StormC-FAQ-english

COLLABORATORS

	TITLE : StormC-FAQ-english		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 8, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

StormC-FAQ-english

1.1 StormC-FAQ

```
                    StormC-FAQ
    Stand: 28 May 1999


                How do I get support?

                Why does the linker generate the error 'Symbol_exit not defined'  ↩
                    ('Symbol

                _exit nicht definiert') when you link as a shared library?

                Why is the 'storm.lib' library so big and why is there - in  ↩
                    contrary to

                SAS/C - only one library?

                Why are other compilers faster than StormC?

                How can a program like 'Hello World' be compiled, so that it is  ↩
                    really

                small?

                Why is a program like 'Hello World' greater than one KByte?

                Why doesn't have StormC a 'global optimizer'?
                    This FAQ is Copyright by Haage&Partner.
    News and most recent FAQs at: http://www.haage-partner.com

    Created on 28 May 1999 by Fiasco written by Nils Bandener
    and by a magic ARexx script written by Martin Steigerwald.
```

1.2 StormC-FAQ

```
 How do I get support?
```

You can get support most easily and quickly using the Internet. To get support from us you have to be a registered user. Please do never send your serial number over the internet unless you use PGP encrypted mail.

```
Info:     http://www.haage-partner.com/sc_e.htm
Support:  http://www.haage-partner.com/sc_sup_e.htm

Mailinglist:   http://www.haage-partner.com/sc_mlist.htm

Email support: <stormc-support@haage-partner.com>
```

1.3 StormC-FAQ

Why does the linker generate the error 'Symbol_exit not defined' ('Symbol _exit nicht definiert') when you link as a shared library?

The shared library calls the ANSI function 'exit()'. This can happen on the one side explicit, if you use the function, or on the other side, implicit, through linker libraries, which use this function. The 'storm.lib' library uses this function when the shared libraries, for example the 'utility.library' are automatically opened.

Normally a shared library is not allowed to use the function 'exit()', because it can't be exited this way. How do you avoid the call?

In this case you are not allowed to use the automatic opening of shared libraries, but you have to open and close the library as described in the StormC documentation. To find out, which libraries are used, you should first use the function void exit() {} in the shared library. Now the library is able to be linked. Use the linker option 'Write map-file' ('Map-Datei schreiben'). The linker creates a file with the suffix '.map'.

Now search all INIT-functions, which have the base name of a shared library (eg. 'INIT_1_UtilityBase'). Now open all these libraries for your own. Don't forget to declare also the base variables (eg. 'UtilityBase') for your own and to remove the own 'exit()' function from your source code.

1.4 StormC-FAQ

Why is the 'storm.lib' library so big and why is there - in contrary to SAS/C - only one library?

StormC supports a newer object format, which is also used in link-libraries. This format is 100% compatible with the older format (upwards and downwards), but the Storm linker and the StormC compiler are able to support more data models with this format in one object file.

So you don't have to choose and choose the wrong library to the chosen compiler options, because the linker chooses the parts out of the big 'storm.lib' which are necessary for the taken data model. Therefore the 'storm.lib' library is as big as the three libraries for the different data models.

In the future StormC will support different code models and CPU and FPU models, so that the 'storm.lib' library and all other libraries will allow the automatic creation of optimized program code.

1.5 StormC-FAQ

Why are other compilers faster than StormC?

StormC creates clean, optimized code with optimal support of register variables and various optimizations. Besides StormC is ready to use the PowerPC and therefore doesn't use fast (but susceptible to errors) assembler parts. This is why StormC isn't that fast like other compilers (eg. MaxonC++). We are however working on a special optimization stage for the development phase of a project with even shorter compilation times.

1.6 StormC-FAQ

How can a program like 'Hello World' be compiled, so that it is really small?

If you do not need floating point datas and the buffering of the Amiga DOS is enough for you, you can use the Amiga DOS for outputs directly. The functions 'VPrintf' and 'VFPrintf' are here for these purposes - to output datas to an Amiga DOS file. But these functions aren't 100% compatible with ANSI C. Another possibility is the renunciation of the automatic opening and closing of libraries, because these functions deliver a comfortable output mechanism for error handling, regardless if the program has been started from CLI or Workbench, and which version of the operating system has been used.

This luxury isn't necessary for every program. You can use a small startup code (which is written in assembler) and do only the most important things there, for example use the small data model and don't support resident programs.

1.7 StormC-FAQ

Why is a program like 'Hello World' greater than one KByte?

The added library 'storm.lib' is an ANSI C library. For a program like 'Hello World' the 'printf' function has to be linked. That means, that the functions of the library which aren't used in the program are also linked to it, like functions for integer or floating points datas because it isn't obvious which parts are necessary for the 'printf' command. Furthermore all printouts to files are buffered. Therefore even short programs get big.

1.8 StormC-FAQ

Why doesn’t have StormC a ’global optimizer’?

A ’global optimizer’ optimizes a program with a look over the whole function, not only over single instructions or terms. Therefore a global optimizer can take a term, which is used in a loop - but in each step of the loop he has to create the same result - and put it in a place before the loop.

StormC doesn’t have this kind of optimization yet, but has been designed in such a way, that it can be implemented in a forthcoming version. But StormC has some optimizations included already now, which are normally part of an global optimizer. For example, the use of the global CPU and FPU registers in the higher optimization stages. The registers are distributed in the whole function optimally, with regard to all function calls and assignments to variables.