

StormC

COLLABORATORS

	<i>TITLE :</i> StormC		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		August 24, 2022	

REVISION HISTORY

<i>NUMBER</i>	<i>DATE</i>	<i>DESCRIPTION</i>	<i>NAME</i>

Contents

1	StormC	1
1.1	StormC.guide	1
1.2	StormC.guide/ST_Order	2
1.3	StormC.guide/ST_CRIGHT	3
1.4	StormC.guide/ST_Lizenz	3
1.5	StormC.guide/ST_Welcome	4
1.6	StormC.guide/ST_Maschine	6
1.7	StormC.guide/ST_Install	6
1.8	StormC.guide/ST_Problem	7
1.9	StormC.guide/ST_Tutorial	8
1.10	StormC.guide/ST_Start	8
1.11	StormC.guide/ST_Project	9
1.12	StormC.guide/ST_Make	9
1.13	StormC.guide/ST_Source	9
1.14	StormC.guide / ST_Compile	10
1.15	StormC.guide / ST_Start	11
1.16	StormC.guide/ST_Debug	12
1.17	StormC.guide / ST_Sektion	13
1.18	StormC.guide / ST_Owns	15
1.19	StormC.guide / ST_Referenz	18

Chapter 1

StormC

1.1 StormC.guide

StormC Preview

Software and documentation

(c) 1995 / 96 by HAAGE & PARTNER COMPUTER Ltd.

Table of contents

License agreement

Chapter 1
Welcome to a new era

Chapter 2
Demands

Chapter 3
Installation

Chapter 4
What to do in the case of "not solvable" problems

Chapter 5
Tutorial

Chapter 6
Your first program

Chapter 7
Generating a new project

Chapter 8
Make and dependency of modules

Chapter 9
Editing the source

Chapter 10	Compiling
Chapter 11	Starting a translated program
Chapter 12	The debugger
Chapter 13	Sections of a project
Chapter 14	Peculiarities of StormC
Chapter 15	Menu commands
	Copyrights and orders
	Order formula

1.2 StormC.guide/ST_Order

Please print the enclosed form on your printer.

Please cross the desired products and fax or transmit us the completely filled form.

Our address:

HAAGE & PARTNER COMPUTER Ltd.
PO box 80

61191 Rosbach v.d.H.

Fax: 06007 / 7543

Order (please mark the desired item(s))

* Yes, I order the complete version of StormC at a price of 598, - DM

* I order the Cross-Upgrade on my old

Compiler system: _____

at a price of 398, - DM

If you do not live in Germany you have to pay in advance plus 20,- DM for shipping.

First name: _____

Name: _____

Street: _____

Zip code: _____ Town: _____

Country: _____

Telephone: _____

E-mail: _____

Per enclosed cash or advance-check

1.3 StormC.guide/ST_CRIGHT

Copyrights and trademarks:

Commodore and Amiga are registered trademarks of ESCOM Inc.

SAS and SAS / C are registered trademarks of the SAS institute.

Amiga, AmigaDOS, Kickstart and Workbench are trademarks of ESCOM Inc.

The designation of products, which are not from the HAAGE & PARTNER COMPUTER Ltd., serves exclusively information purposes and presents no trademark abuse.

1.4 StormC.guide/ST_Lizenz

Licensee agreements

1 In general

(1) Object of this contract is the use for computer programs of the HAAGE & PARTNER COMPUTER Ltd., for the manual as well as for other pertinent, written material, subsequently summing up as product.

(2) The HAAGE & PARTNER COMPUTER Ltd. and/or the licensee indicated in the product are owners of all rights of the products and the trademarks.

2 Right of usufruct

(1) The buyer does receive a non-transferable, non-exclusive right, to use the acquired product on a single computer.

(2) In addition the user can produce one copy for security only.

(3) The buyer is not legitimated, to expel the acquired product, to rent, to offer sublicenses or to put these in other ways at the disposal of other persons. ↔

(4) It is forbidden, to change the product, to modify or to re-assemble it. This prohibition counts also for the translating, changing, re-engineering and re-use of parts.

3 Warranty

(1) The HAAGE & PARTNER COMPUTER Ltd. guarantees, that to the point in time of the delivery the data carriers are physically free by material- and manufacturing defects and the product can be used like it is described in the documentation.

(2) Defects of the delivered product are removed from the supplier within a warranty period of six months from delivery. This happens after choice of the supplier through free replacement or in form of an update.

(3) The HAAGE & PARTNER COMPUTER Ltd. does not guarantee that the product is suitable for the task anticipated by the customer.

For possibly appearing damage the HAAGE & PARTNER COMPUTER Ltd. does not take over any responsibility.

(4) The user knows, that after the present status of the technology it is not possible to manufacture faultless software.

4 Other

(1) In this contract all rights and responsibilities of the contracting parties are regulated. Other agreements do not exist. Changes are only accepted in written form and at reference on this contract and have to be signed mutual.

(2) The jurisdiction for all quarrels from this contract is the responsible court at the headquarters of the HAAGE & PARTNER COMPUTER Ltd.

(3) If any single determinations of these conditions is not according to the law or lose their lawness through a later circumstance, or should appear a gap in these conditions, this should not touch the remaining determinations.

In lieu of an ineffective term of the contract or for the completion of the gap an appropriate regulation should be formulated which comes next the statement, which the contracting parties had in mind as they worked out this contract.

(4) Each injury of this licence agreement or by originator and trademark right-wingers is pursued under civil law.

(5) Through installation of the software these license agreements are accepted.

(6) If you do not agree with this license agreement you have to give back the product immediately to the supplier.

September 1995

1.5 StormC.guide/ST_Welcome

Welcome to a new era of Amiga programming.

With the enclosed preview of our brand-new compiler system you will get to know the abilities of a progressive programming language.

In a so-called integrated environment you will find everything, which you require for programming. Heart and center is the project management, by which all components are called up and are provided with data.

The project management is not simply a better MAKE, but an administration for all your program modules. Among them e.g. sources, object libraries, documentation, ARexx scripts, pictures and resources are managed. All compiler, editor and project options are managed here too.

If you have got the impression now, that everything to serve is certainly much to complicated, I can quiet you. Please look at the next pages, where the first example is described and you will realize very quickly, that everything can be done very easy and intuitive.

A further component of the system is the editor with its particular ability

to emphasise key-words and syntax characteristics colourfully. With this text colouring you can read your program much easier, because you will see the particular structuring better. Apart from this it helps you to avoid errors at the editing of your sources. As soon as a key-word or an Amiga function is entered, the word is marked colourfully and you know you did it right.

Now I will introduce to you the extraordinary debugger. Extraordinarily because there is not difference, whether the editor or the debugger is running. The debugger uses the abilities of the editor, this means that the debugger uses the editor window for its output. So you can watch the source, set breakpoints, look for functions and variables and so on with the ease of using the editor. The structuring and the colouring of the source are helping you to do your debugging job.

When we planned the features of StormC we had some ideas in mind that we want to realise in the future. We are planning to reinsert the changes that you have done in the debugger directly into the running program. You do not have to leave the debugger, nor to compile and re-start the program. As you see, this will make software development much easier and much more efficient.

Another big help for the debugger is our "RunShell". With it it is possible, to localise the typical errors on the OS programming very quick. One example for errors, which are made again and again, is the use of the functions AllocMem() and FreeMem(). To allocate memory is a simple thing, but to give it back to the OS seems to be a big problem for many programmers. Either they forget to free all memory or the block is too large or too small, which mostly ends with a CPU exception. The RunShell records all important things that deal with system routines resources. So it knows exactly, when there is a function call with too much, too low or faulty parameters.

Another big advantage of the RunShell is the possibility to start the debugger at any time during the runtime of the program. You do not have to decide this before you start the program. If you want to debug the running program, just start the debugger from the RunShell. It is that easy.

Now, let us talk about the real important thing of our development system - the compiler. The idea of the object oriented programming is up-to-date, but not everyone is developing in C++. So we decided to make a compiler for both parties.

The traditional programmers will use our very fast and compatible ANSI C compiler. They can switch to the object oriented programming with C++ at any time, completely or partially. StormC is their tool for the future.

The others will use our outstanding C++ compiler. StormC offers the C++ implementation according to the implementation of Bjarne Stroustrup and it supports the expanded AT&T standard version 3.

The compiler generates code for all Motorola 680x0 CPUs including the 68060 CPU. The fundamentally outstanding speed of the compiler is accelerated through precompiled. The integrated linker processes all current library formats (SAS/C, MaxonC++, ...) and is thereby one of the fastest on the Amiga.

StormC is suitable for all programming projects e.g. administration-, graphic-, music- and game programs. For all these projects StormC should be your first choice. The existing preview version of StormC helps you with the decision for your future compiler system. Therefore we do not offer a self-running demo version ↔

because you certainly want to test your own sources and the possibilities of the whole system.

The preview version of StormC has no limits in source length or things like that. But you should always keep in mind that it is a preview, not the final version, which will be released in January 96. There will be some changes to the GUI and the functionality and of course to the stability of the whole system.

If you find a function, that does not work or does not work the way you would like it, please feel free to send us a message.

Our address:

HAAGE & PARTNER COMPUTER Ltd.
PO Box 80
61191 Rosbach
Germany

Phone: ++49 - 6007 - 93 00 50
Fax : ++49 - 6007 - 75 43

Compuserve: 100654,3133
Internet: 100654.3133@compuserve.com
WWW: <http://home.pages.de/-haage>
WWW: http://ourworld.compuserve.com/homepages/-haage_partner

1.6 StormC.guide/ST_Maschine

Configurations

In the following list you will find the minimally configuration for using StormC:

- Amiga with hard disk
- Kickstart and Workbench 2.0 (v37)
- 3 MB RAM
- 5 MB hard disk space

With this computer system you can start programming with StormC, but the project size is limited. Furthermore all debugger things can not be used. For this you need at least a 68030 CPU with MMU and more RAM.

A really good configuration for StormC is the following:

- Amiga with 68030 including MMU
- Kickstart and Workbench 3.0 (better 3.1)
- 8 MB RAM
- 30 MB hard disk space

As a rule always remember: THE MORE THE BETTER!

1.7 StormC.guide/ST_Install

Installation

For the installation on your hard disk the "Installer" of AT/Commodore will be used. This tool has prevailed in the meantime as standard.

Please insert the first disk of the preview into your disk drive and double click on the disk icon. Before you start the installation, please read the "Readme" file in the root directory of the disk. Herein are important innovations and hints that could not be included in the manual any more.

After this, double click the icon "Install StormC to HD". Please wait a moment, ←
until
the installer and the script is loaded. Now follow the orders of the installation program. If you are not sure what to do, simply click the "Help" button to get further information.

If the installation worked successfully you will receive a corresponding message.

If the installation was not successful, please repeat it while writing a LOG FILE. The option "Log all actions to: Log File" can be selected in the option window, just at the beginning of the installation procedure. After the (←
unsuccessful)
installation you can read this log file to see what had happened during the installation. Remove the cause of the problem and start the installation again.

1.8 StormC.guide/ST_Problem

What do in the case of "not solvable" problems

If you have problems with StormC during the installation or later, please feel ←
free
to contact us.

Here you can reach us:

HAAGE & PARTNER COMPUTER Ltd.
PO Box 80
61191 Rosbach
Germany

Phone: ++49 - 6007 - 93 00 50

(on workdays between 9:00 and 17:00 (MEZ))

Fax : ++49 - 6007 - 75 43

Compuserve: 100654,3133

Internet: 100654.3133@compuserve.com

WWW: http://ourworld.compuserve.com/homepages/haage_partner

1.9 StormC.guide/ST_Tutorial

Tutorial

In the following part of the manual you will learn everything about the handling of StormC. Using impressive examples the functions of the compiler system will be shown. You learn the main things about the project management, how to edit sources and how to start the compiler. A step by step example will show you the work with the debugger.

With this tutorial you will get an impression of the power of the StormC development system. And you will never work with another one.

1.10 StormC.guide/ST_Start

Your first program

Now you will see how to start a new project, how to edit the source and how to compile and start it.

Please start StormC through a double click on the program icon. You will find the program in the drawer you installed StormC into.

During the start up you will see a welcome message, which is to be seen on the screen as long as components of StormC are loaded. After this you will see a vertical toolbar on top of your screen. It contains the most important functions of StormC which can be chosen very quickly.

The icon bar provides the following functions:

New Text
Load Text
Store Text

New Project
Load Project
Store Project

Start Compiler (Make)
Execute Program (Make and Run)
Start Debugger (Make, Run and Debug)

We will start - of course - with the one and only typical example: "HELLO WORLD".

Hello World-source code

1.11 StormC.guide/ST_Project

Generating of a new project

Please click the icon "new project". A new project window is indicated.

What is a project?

A project is the summary of all related parts of your program e.g.: C -, C++ -, assembler sources, headers, object files, link libraries, documentation, graphics, pictures and other resources. Through the separation in various sections you will always keep the overview. The project management is also a graphically oriented Make.

1.12 StormC.guide/ST_Make

Make and dependency of modules

On each compiler pass the dependence between ".o", ".h", ".ass", ".asm", ".i", ".c" and of course ".cc" and ".cpp" files will be checked by the project management. So the project management knows by itself, that a C source must be compiled newly, if one ".h" header which was included in the ".c" source was changed.

At the click on the icons "Make" or "Run" all dependence are examined. The Make decides, which module of the program must be compiled newly. "Run" differs to "Make" solely thereby, that at successful compilation the program is started automatically.

Project store and the making of a new directory

Next you should store your project into a new directory. Click on the "Save Project" icon. Choose the directory "StormC:" and enter the path and the file name "Hello World/Hello World". The ending ".\u2197" is appended automatically and indicates that this is a StormC project. You can enter this ending manually by typing <ALT> + <P>.

You will certainly wonder, why an empty project should be stored; even if the project has still no content. We recommend to do this, because now the paths of the sources and resources can use it as an relative path and it is easier to handle the whole project. Moreover this path will be the default in the file requester.

1.13 StormC.guide/ST_Source

Editing the source

Now we can start with our project. Please open a new edit window with a click on the icon "New Source".

Before you begin to enter the source, I will introduce the main elements of the editor which are in the upper tool bar. ↔

E = toggles line end marking on/off
T = toggles tab marking on/off
S = toggles space marking on/off
I = toggles indenting on/off
O = toggles overwriting on/off
R = toggles write protection on/off

The next field shows if you have done any changes to the text.

At the right side of the toolbar is the rows and columns display.

Please type the following text now:

```
/ * Hello World
   demo of the preview tutorial * /

# include <stdio.h>

void main (void)

printf ( "Hello World\n");
}
```

Store this program with the name "Hello World.c" in the directory "Hello World".

Choose the menu item "Project / Add Window". Now the file name appears in the source section of the project window. The project management looks after the endings to select the section of the particular file.

Before you start the compiler you should indicate a file name for your program. Otherwise the file name "a.out" is automatically chosen.

Select the menu "Projekt" and the item "Select name of the program". Pay attention that the project window is the active window.

1.14 StormC.guide / ST_Compile

Compiling

Click on the project icon "Make". The error window of the compiler is opened and the translation of the source starts. During this some messages are print in the error window. ↔

If an error occurs during the compilation a description will be printed which contains the (possible) matter of the error and the line number of the source code.

With a double click you will get back to the editor just onto the line with the (possible) error. Make your corrections and start the compiling again.

1.15 StormC.guide / ST_Start

Starting a program

In the case of a successful compiling and linking the button "Start" becomes accessible. Click on it or the "RUN" icon in the tool bar to start your program.

If you choose "Run" instead of "Make" the project management first examines, whether all modules are already compiled. If this is not the case, it now starts the compiler. After the successful translation, the program is started automatically.

The RunShell

Starting a program out of the project management is something more special, so I am going to tell you about it.

You have certainly noticed that when you started a program that another window is opened. This one is called the "RunShell".

Naturally it is also possible to simply start the program, but if your program is under development you often wish to have more control over the running program. With the RunShell of StormC it is possible to debug the program after running it. If an error appears that normally causes a CPU exception the RunShell takes control (in most cases) and gives you the opportunity to look at the error in the source.

A further important characteristic of the RunShell is the "Resource Tracking". To allow this all system functions that have to do with resource handling (AllocMem, OpenWindow, Open, ...) are recorded. When the program is finished, the "Resource Tracking" checks if there are resources which are not freed or freed too often. You can now go directly to the source and make the changes immediately.

Furthermore the RunShell offers the possibility to send the signals Ctrl-C, Ctrl-D, Ctrl-E, Ctrl-F, which can normally only be sent by the Shell (CLI), from which the program was started.

The priority of the program can be manipulated in steps between -128 to +127.

The "Pause" button stops the active program. "Pause" is a toggle button. A further click and the program runs again.

With a click on the "Kill" button the program will terminate. All recorded resources are released (storage and signals are released; screens, windows, requester and files are closed). By that reason there will be no dead programs

in your system. Through inadvertently programmed endless loops emerge easily such dead-end programs. This method will save you a lot of time, because it is much faster than rebooting your Amiga.

In our example the program is processed so quickly that the RunShell window will be closed shortly after its opening. The next example will show you more of the RunShell and the work with the debugger.

The output window ("Hello World") that is still open on your desktop is a normal console window. This type of window is opened automatically if your program uses standard input/output routines e.g. the function "printf". To close it simply click on the close gadget of the window.

1.16 StormC.guide/ST_Debug

The debugger

The debugger is required for the fast search of errors in your program. It offers the possibility to define breakpoints in your programs and to observe changes of variables, structures and classes at these places. In this way errors can be encircled and you will be able to remove them quickly.

To demonstrate the work of the debugger, you should load an existing project and compile it.

In the directory "examples" you will find the file "Colorwheel". Open the file on the Workbench with a double click on the icon. The project will be loaded and indicated. Choose the menu "Settings" and the item "Project". Click on the upper cycle gadget until "C/C++ options" appears. As you see, "large debug files" is a preset.

Start the compiler with a click on the debugger icon. The system checks now \leftrightarrow whether there are debugger files for all modules or if they must be compiled. This is the same procedure as if you hit the RUN icon. After linking the program will be \leftrightarrow started directly in the debug mode.

According to the preset the module, active variables and monitored variable window will be opened.

Furthermore the sources of the module which contains the main function is opened and indicates its sources.

You will see, that the output in the editor window changed a bit. The first column of the text is shift to the right, so that an additional column can be indicated which shows the breakpoints. The breakpoint fields are only in the lines at which the program can be stopped. If you click with the mouse on one of these points \leftrightarrow they are double crossed. This indicates an active breakpoint.

Please set a breakpoint directly after the "OpenScreen" call.

If the window of the "current variables" is not open please do so by selecting the menu item "Windows / Current variables".

Click on the icon "Go up to the next Breakpoint" on the RunShell button bar.

Since you have set the breakpoint directly after the function call "OpenScreen", the program is executed until here. A new screen is opened and the program is stopped now.

The next function "GetRGB32" provides an array of unsigned long characters with data. Now we want to examine this more deeply.

Here are some explanations of the buttons of the inspect window:

Q = show corresponding place in the source code
F = show member function of a class type
I = inspect
H = show HEX editor
W = take over the variable to the inspect window
Cast = temporarily change type of variable
Low/High = borders of the array display

Now it is necessary to put the array variable colortable to the inspect window.

Simply choose in the window "current variables" the variable "colortable" with the mouse and click on the symbol "I" at the upper edge of the window.

Before this I give you some explanations of the buttons of the inspect window:

I = inspect
P = previous
H = show HEX editor
W = take over the variable to the inspect window
Cast = temporarily change type of variable
Low/High = borders of the array display

Next you execute three single steps and observe how the content in the inspect window ↔ changes. Click the symbol "go one step" on the RunShell twice. The function " ↔ GetRGB32" loads the values of the screen view structure into the array "colortable".

You see how simple it is, to monitor variables with the inspect window.

Now set further breakpoints and "play" with the debugger and its functions. You will ascertain very quickly how simple the system is to be served.

To end the debugger terminate the program started in the debugger mode.

All debug windows will be closed automatically and the RunShell will be terminated ↔

1.17 StormC.guide / ST_Sektion

Sections of a project

The sections of a project are created automatically through the file ending and at documents in addition through the file names. If you add a ".c" file to a new project a new section emerges with the name "sources" counteracting the corresponding files. At further adding of sources the section is expanded solely.

Following sections can be created by now:

Sources

- ".c"
- ".cc"
- ".ccp"
- ".c++"
- ".cpp"

Headers

- ".h"
- ".hh"
- ".hhp"
- ".h++"
- ".hpp"

ASM sources

- ".asm"
- ".ate"
- ".s"

ASM headers

- ".i"

Documentation

- ".dok"
- ".doc"
- ".txt"
- ".readme"
- "read me"
- "liesmich"
- "readme"
- "read me"
- "read.me"
- "lies.mich"

ARexx

- ".rexx"

Others

- "*"

Projects

- ".fl"

Amiga Guide

- ".guide"

Program

1.18 StormC.guide / ST_Owns

Peculiarities of StormC

Despite the standards ANSI C specifications each compiler has his own ←
peculiarities.

These specialities are introduced with "#pragma". #pragma lines are, exact as # ←
include,
interpreted and executed by the pre-processor.

Modes of the compiler

#pragma -

is a non-standard feature that shifts the compiler to the ANSI-C mode.

#pragma +

causes the compiler to translate the source in the C++ mode.

Chip- and Fastram

The architecture of the Amiga is sometimes a bit unorthodox. So there are two or ←
more
different classes of RAM.

Normally the programmer is only interested in the answer to "Chip or Fast-RAM",
because he has to care for graphic data to be filled in the Chip memory. StormC
therefor offers the pragmas "chip" and "fast" .

All after the line

#pragma chip

declared static data are loaded into the Chip-RAM.

#pragma fast

switches into normal mode, in which data is put somewhere, preferable in the
Fast-RAM.

Breakpoint control

The breakpoint control has nothing to do with the Breakpoints in the debugger,
rather with the possibility of the compiler to create programs which can be ←
interrupted
by the user. That is pleasant during the program development (especially if you
have programmed an endless loop) and for the user of a program, because he can ←
abort it.

The breakpoint control is testing the signal bit, which is set by the keys <CTRL> ←
+ <C>
or the CLI command "break" on every location with a minimum of one time per loop. ←
This
costs of course a lot of time. And sometimes it is not desirable that a certain ←
action

is interrupted. For that reason the interrupt possibility is optional. You will ←
 find
 the switch "interruptible" in the options window.

However if you want to use the interrupt option in some routines but not in all,
 use the pragma "break".

```
#pragma break +
```

Insert the breakpoint control.

```
# pragma break -
```

Ends the breakpoint control.

If you press <CTRL> + <C> (interrupt option is on) this corresponds to "exit(900) ←
 ".

All files are closed and all memories and other resources are freed. Functions of
 "atexit" will be executed. Naturally such a cancellation is a low-level construct,
 so that no destructors are called.

OS calls

The Amiga OS (operating system) functions are called with #pragma amicall.

Such a declaration exists in the essential of four parts:

- name of the base variable.
- offset as a positive integer.
- name of the function (must be declared already); for reasons
 of the definiteness these names may not be overloaded
- list of parameters (represented through a corresponding amount of
 register names in round brackets)

An example:

```
#pragma amicall(Sysbase, 0x11a, AddTask (a1,a2,a3))
#pragma amicall(Sysbase, 0x120, RemTask (a1))
#pragma amicall (Sysbase, 0x126, FindTask (a1))
```

Normally you will never write such declarations by your own, since everything is
 included with the Amiga libraries.

Joining lines

In C and in C++ it is absolutely alike where lines end. At the pre-processor this
 is pertinent, for each instruction must fit in exactly one line. Sooner or later
 you may come to the point, e.g. through an extensive macro definition, where you
 get a very long line. For this case there is the backslash. If a "\" stands at
 the end of the line this is joined with the following, e.g.:

```
# define SOMETHING \  

47081115
```

This is a valid macro definition, for "47081115" is pulled in the preceding line.

Predefined symbols

The pre-processor owns many predefined macros. Some are ANSI C standard, others are ←
 element of C++ or particular peculiarities of StormC. These macros can not become ←
 re-defined.

`__COMPMODE__`

is defined in StormC as the integer constant "0" in the C mode and under C++ with ←
 "1".

`__cplusplus`

In StormC the macros "`__STDC__`" is defined in C as well as in the C++ mode. If you ←
 want to check whether the compiling will be done in C++ mode, this must be made ←
 with
 the macro "`__cplusplus`".

`__DATE__`

The macros `__DATE__` delivers the date of the compilation. This is very useful if ←
 you want to furnish a program with an unique version number:

```
#include <stream.h>
void main ()
{ cout << "version 1.1 from " __DATE__, " __TIME__" clock\n; }
```

The date is delivered in the form month - day - year , e.g. "Feb 08 1996"; the ←
 time
 is set quite normal in the form "hours:minutes:seconds".

`__FILE__`

This macros contains the name of the current source file as a string variable, e.g ←
 .:

```
# include <stream.h>
void main ()
{ cout << "That stands in line " << __LINE__ << " in the file " __FILE__ ".\n"; }
```

The value of the macros `__FILE__` is a constant character string and can be joined ←
 with leading or following strings.

`__LINE__`

The macro `__LINE__` delivers the line number, in which it is used, as a decimal " ←
 int"
 constant.

`__STDC__`

This macro delivers the numerical value 1 if the compiler is compatible to the ←
 ANSI C standard. Otherwise is not defined.

`__STORMC__`

This macro gives you the name of the compiler and the version number.

__TIME__


(see __DATE__)


1.19 StormC.guide / ST_Referenz

Menu commands

Project

New A-N

At active icon or project window a new project will be opened. This corresponds to  a click on the icon "new project".

If an edit window is active, a new edit window will be opened. That corresponds to  a click on the icon "new text".

Open... A-O

The standard (ASL library) file selector will open to read a file. If the icon or project window is active a project will be opened. If the editor window is open a text will be loaded. You can choose Open... with the icon bar as well.

Save A-S

At active project window the project is saved. This corresponds to a click on the icon "Save Project".

If an edit window is active, the text is saved. That corresponds to a click on the icon "Save Text".

Save as...

The standard file selector opens. You have the possibility to indicate a file name for your project or your text.

Depending on whether the project window or an edit window is active, you can save either the project or the text. The icon bar offers two icons to save the sources and projects.

Save as project pattern

This menu item is only accessible if the project window is active. The project pattern is loaded if you choose a new project. You can set the options which are used at each of your next projects.

All options of a project (C/C++ environment, C/C++ pre-processor, C/C++ options, C/C++ warnings, linker options and program start) and of course all sections of the project management.

Save all

With a click on the menu item of all source and projects which are not yet stored will be saved. If no file names are indicated for sources or projects the file selector is opened for each missing file name.

Add files...

These menu item is only accessible if a project window is activated. A file can be selected from the file selector, which is taken into the project manager. ←

According to file ending different sections are created and the file is placed at the corresponding position in the project.

Add window

This menu item is accessible when there is a project and the edit window is active ←

With a click the file of the active edit window is placed in the corresponding project section. In contrast to "Add files" the file selector does not appear.

Choose program name...

Of course each program requires a name. As a program consists of many modules, ← this can not be done automatically. With the help of the file requester you can enter ← the program name and choose a different location.

Close A-K

Depending on whether a project or edit window is active, either the project or the text is closed. If the project or the text was not stored before, a safety request appears and offers you the possibility to do this now.

The same will happen if you click on the close gadget of the window.

Info

In the indicated dialogue you will find information to the product, copyright, our current telephone, fax number and our email address.

Quit A-Q

StormC will be terminated. If sources or projects are not stored by now, you will get a message to do so.

Edit

Mark A-B

The menu item is only eligible at activated edit window. It switches the marking mode of the editor.

Cut A-X

The menu item is only eligible at activated edit window. The marked text area is copied to the clipboard and deleted from the active text.

Copy A-C

The menu item is only eligible at activated edit window. Unlike when cutting the text, now it is not deleted, but copied to the clipboard.

Paste A-V

The menu item is only eligible at activated edit window. Thereby the text located in the clipboard is inserted at the cursor position in the active text.

Delete

At active edit window the marked area is removed from the text. The deleted text is not copied into the clipboard. This can be cancelled however with "Undo".

At active project window the module marked in a section is removed from the project. ↔

"Undo" is not possible!

Undo A-Z

The menu item is only eligible at activated edit window. "Undo" takes back the finally executed editor function.

Redo A-R

The menu item is only eligible at activated edit window. With "Redo" you can take back the finally executed "Undo".

Find & Replace... A-F

Find & replace is eligible of course only at active editor window. In the appearing ↔

dialogue you can enter the search key and the replace word can be entered.

The function can be used for searching as well. Do not enter a substituting notion and click on the symbol "Find".

In addition the direction can be selected indicated from the cursor position.

With the cycle menu you can define the options more exactly. Moreover you can ignore upper and lower cases and accents.

With the three symbols at the lower edge of the dialogue you execute the instructions in different ways.

"Find" searches solely the find string

"Replace" replaces the find string with the replace string singularly.

"Replace all" replaces all found strings through the replace string

Find next A - " . "

The menu can only be selected at active editor window.

"Find next" repeats the finally made find access without opening the dialogue again.

Replace next A - "-"

The menu can only be selected at active editor window.

"Replace next" repeats the finally made replace access without opening the dialogue ↔

before.

Compile

Compile...

The menu item "compile. . ." is available if an entry is marked in the project ↔ section

of the sources. It can be also be chosen, if the source indicated in the active editor window is found in the active project.

With this function you can compile individual modules.

Make... A-M

The menu item is eligible only at activated project or error window. It can also be chosen, if the source indicated in the active editor window is found in the active project.

With "Make" all modules are compiled, which were changed since the last compiling. Thereby dependence are considered between program and header files.

To proceed this, you can also click on the icon "Make" .

Run...

The menu item is eligible only at activated project or error window. It can also be chosen, if the source indicated in the active editor window is found in the active project.

If there is already a up-to-date version of the program, this will be started. Otherwise a "Make" will be executed that compiles all changed modules. After this the new program will be started.

To proceed this, you can also click on the icon "Run" .

Debug... A-D

The menu item is eligible only at activated project or error window. It can also be chosen, if the source indicated in the active editor window is found in the active project.

"Debug..." does the same as "Run" plus starting the debugger. The program counter is set on the first function (main) and the program is stopped at this position.

To proceed this, you can also click on the icon "Debug" .

Touch

The menu item is eligible only at activated project window.

If you "Touch" an entry in the source section of the project manager it will be marked as "changed". At the next starting by "Make" this source is compiled guaranteed newly.

Touch all

The menu item is eligible only at activated project window.

Herewith all modules of the source section are marked "changed". At the next starting by "Make" this source is compiled guaranteed newly.

Save program as...

The menu item is only eligible at activated project or error window.

After linking a program successfully it is automatically saved. If you did not give a name to the program it will be saved under the default "a.out" in the directory, in which the project is stored. With "Save program as..." you now have the possibility to store a copy of the created program with another name at another place on your hard disk.

Windows

Error window...

The menu item is eligible only at activated project window.

It opens the error window.

Modules...

The menu item is only eligible if the debugger is active. It opens the window of ↔ the modules, for which information can be outputted. In principle this equals the list ↔ of entries that appears in the project window under sources, if for each source a ↔ debug file was created. Instead of the source names you will find the file names of the object modules.

Current variables...

The menu item is only eligible if the debugger is active. It opens the window of ↔ the current variables.

Monitored variables...

The menu item is only eligible if the debugger is active. It opens the window of the monitored variables.

Options

Project...

These options are only eligible if the corresponding project or error window are active.

Include paths and pre-compiled headers

From the include files the pre-processor reads its definitions for e.g. the ↔ standard functions. In your source you use the pre-processor instruction #include for this. As you certainly know, there are two possibilities to do this:

If one includes the file names after the "#include" "like this" in citations the pre-processor (the software module, which processes the file before the compiler) looks for them in the working directory.

If you put them <like this> in brackets this represents file names of standard include files and they are looked for in pretended directories. In the Listview you can choose several directories for the standard include files.

"Copy to:" gives you the option to speed up compiling. You can copy the files to ↔ the RAM DISK, because this makes the compiler very fast, but of course you need some memory for this.

Another way to speed up the compiling dramatically and even to save RAM is to use pre-compiled header files. To tell this to the compiler, you have to mark where the header file ends and your program starts with "#pragma header". A simple way is to write the "#includes" of all header files that are not yours or never changed (e.g. OS includes) to the first position of each module. Then the line "#pragma header" follows. Hereafter you can go ahead with your own header files and the rest of your program.

The pragma instruction has no effect as long as you do not select "Write header ←
file"

and re-compile the changed headers. As soon as the compiler reaches the pragma instruction the pre-compiled header files are written under the indicated name into the corresponding directory.

Before you start the compiler the next time you simply switch to "read header ←
files".

The compiler reads the pre-compiled header file, searches the instruction "#pragma header" and starts its translation.

You will reach the next option dialogue by cycling through the cycle menu on the upper edge of the window.

Pre-processor

Here you can select the pre-processor warnings and predefine the pre-processor ←
symbols.

The items of the pre-processor warnings are easy to configure, but the definitions should be explained a bit.

Each entry you make in this Listview will be the same as writing it with a "# ←
define"

in front to every source file. With this you can make global definitions as e.g. "Debug" and as a token "True" or "False".

Generating code and debugging

Choose the sort of the source (ANSI C or C++) and in the case of C++ the ←
processing
of templates and the use of the exception handling.

The next cycle menu gives you the opportunity to choose debug output or not. If ←
you

want debug output the compiler generates additional files with the ending ".debug" and ".link". These files are required by the debugger, to know the relation ←
between
sources and program code.

If you want to work with a symbolic debugger/disassembler you have the possibility to add a symbol hunk to the program.

You can even get assembler sources. The compiler creates additional ".o" and ".s" files. They contain alternatively assembler source mixed with c-source.

If you choose "interrupt code generation" the compiler generates a query of <CTRL> + <C> in every loop.

You should turn on "32 bit multiplication" if you create code for the 68000 ←
because

then the correct library calls are used to the long word division and ←
multiplication.

If you generate code for higher processors this switch is insignificant.

"Optimise code" makes the program more compact and mostly faster.

The next cycle gadget selects the processor type. Please think of the downward incompatibility: if you create for the 68060 this program will not run on a normal Amiga 2000 any more.

Next you can choose between generating FPU code or calling the system libraries for mathematical calculations. ↔

The last cycle menu shifts between large and small data model.

Warnings

StormC differs eight warnings, which you can switch on and off according to your individual needs.

Linker settings

Similarly like for include files at the pre-processor options the path for the linker libraries can be set. ↔

The following cycle menu has three options:

"Link program" links the compiled program with the libraries.

"Do not link" does not start the linker.

"Link without startup code" is used for shared libraries or device drivers. You can not start such a program from CLI or Workbench. ↔

If an error occurs during the compilation you can decide whether to link the program or not. ↔

Running

If you run a program out of the integrated environment of StormC you will get the possibility to indicate arguments and the stacksize of the program.

Load settings...

You can load a debugger settings file with the ending "RUN".

This item is only available if the debugger is active.

Save settings

You can store the complete debugger configuration (which windows are opened, which position). The "StormSettings.Run" is saved to the home of StormC.

This item is only available if the debugger is active.

Save settings as...

You can store the complete debugger configuration as mentioned above, but you can choose a different name and location. This item is only available if the debugger is active. ↔

active. ↔