StormC

COLLABORATORS			
	TITLE : StormC		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 19, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

StormC

1.1 StormC.guide

```
StormC Preview

Software and documentation
(c) 1995 / 96 by HAAGE & PARTNER COMPUTER GmbH


Table of contents

 License agreement

 Chapter  1   Welcome to a new era
 Chapter  2   Requirements
 Chapter  3   Installation
 Chapter  4   What to do in the case of "insoluble" 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 form
```

1.2 StormC.guide/ST_Order

```
Please print the enclosed form on your printer.

Please check the desired products and fax or send us the completely
```

filled-out form.

Our address:

HAAGE & PARTNER COMPUTER GmbH
PO box 80

61191 Rosbach v.d.H.

Fax: +49 6007 / 7543

Order(please mark the desired item(s))

* Yes, send me the complete version of StormC
at a price of 598,- DM (398,- US$)

* I want to order the Cross-Upgrade from my old

Compiler system: _____

at a price of 398,- DM (269,- US$)

If you do not live in Germany you have to pay in advance plus
20,- DM (15,- US$) 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

trademark abuse.

1.4 StormC.guide/ST_Lizenz

Licensee agreements

1 In general
(1) Object of this contract is the use of computer programs from the HAAGE & PARTNER COMPUTER GmbH, including the manual as well as other pertinent, written material, subsequently summed up as the product.
(2) The HAAGE & PARTNER COMPUTER GmbH. 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 GmbH guarantees, that up to the point in time of delivery the data carriers are physically free of material and manufacturing defects and the product can be used as described in the documentation.
(2) Defects of the delivered product are removed by the supplier within a warranty period of six months from delivery. This happens through free replacement or in the form of an update, at the discretion of the supplier.
(3) The HAAGE & PARTNER COMPUTER GmbH does not guarantee that the product is suitable for the task anticipated by the customer.
The HAAGE & PARTNER COMPUTER GmbH does not take any responsibility for any damage that may be caused.

(4) The user is aware that under the present state of 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 in reference to this contract and have to be signed by both parties.
(2) The jurisdiction for all quarrels over this contract is the court responsible at the seat of HAAGE & PARTNER COMPUTER GmbH
(3) If any single clause of these conditions should be at odds with the law or lose its lawfulness through a later circumstance, or should a gap in these conditions appear, the unaffected terms will remain in effect.
In lieu of an ineffective term of the contract or for the completion of the gap an appropriate agreement should be formulated which best approximates within the bounds of the law the one that the contracting parties had in mind as they agreed on this contract.
(4) Any violation of this license agreement or of copyright and trademark rights

will be prosecuted under civil law.
(5) The installation of the product constitutes an agreement with these license conditions.
(6) If you should not agree with this license agreement you have to return the product to the supplier immediately.

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 you require for programming. Heart and center is the project management facility, from which all other components are invoked and are provided with data. The project manager 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 from here too. If you are under the impression now that controlling all this must be much too complicated , I can set your fears at rest. Please look at the next pages, where the first example is described and you will realize very quickly, that everything can be done very easily and intuitively.

A further component of the system is the editor with its particular ability to emphasise keywords and syntax characteristics colourfully. With this text colouring you can read your program much easier, because you will be better able to see its structure. Apart from this it helps you avoid errors while editing your sources. As soon as a keyword or an Amiga function is entered, the word is marked colourfully and you know you did it right.

Next, allow me to introduce you to the extraordinary debugger. Extraordinary because it makes no 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 one 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 naturally had some ideas in mind that we still plan to add in the future. In future versions we intend to let you integrate changes to the source file that you make from the debugger directly into the running program. You won’t need 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 locate typical errors in OS programming very quickly. One example of the errors that are made again and again, can be best illustrated with the functions AllocMem() and FreeMem(). Allocating memory is easy enough to do, but giving it back to the OS seems to be a big problem for many programmers. Either they forget to free all memory or they get the size of the block wrong, usually

resulting in a CPU exception. The RunShell remembers all important data relating to system resources. Thus it knows exactly when these functions are called too often, not often enough or with incorrect arguments.

Another big advantage of the RunShell is the possibility to start the debugger at any time while running 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 most important part of our development system - the compiler. Object-oriented programming is all the rage. Hardly any software developer programs in ANSI C anymore, at least that is the impression one gets. The truth, however, is quite the opposite. While many programmers use C++ compilers, these are suited just as well for translating ANSI C code.

That is why 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 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 implements C++ according to the design by Bjarne Stroustrup and it supports the extended AT&T 3.0 standard. The compiler generates code for all Motorola 680x0 CPUs including the 68060. The fundamentally outstanding speed of the compiler is accelerated by a large factor through the use of precompiled header files. The integrated linker processes all current library formats (SAS/C, MaxonC++, ...) and is one of the fastest linkers for the Amiga to boot.

StormC is suitable for all programming projects, be they administrative, graphics, music or 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; after all you will certainly want to test it with your own sources and compare the system to your old compiler.

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 GmbH
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

Requirements

In the following list you will find the minimal 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 not all debugger features can 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 (3.1 is even better)
- 8 MB RAM
- 30 MB hard disk space

As a rule always remember: THE MORE THE BETTER!

1.7 StormC.guide/ST_Install

Installation

The standard AT/Commodore "Installer" is used for installing the package on your hard disk. As most Amiga software uses this program you are probably familiar with its operation.

Please insert the first disk of the preview in 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 additions and hints that could not be included in the manual any more.

After this, double click the icon "Install StormC to HD" and wait while the Installer utility and the installation script are loaded. Now follow the instructions of the installation program. If you are not sure what to do, simply click on the "Help" button to get further information.

If the installation was successful 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, right at the beginning of the installation procedure. After the (unsuccessful) installation you can read this log file to find out what went wrong. Remove the cause of the problem and start the installation again.

1.8 StormC.guide/ST_Problem

What do in the case of "insoluble" problems

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

You can reach us at:

HAAGE & PARTNER COMPUTER GmbH
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 operation of StormC. The Features of the compiler system will be shown using convincing examples. You'll learn the basics of the project manager, how to edit source files and how to start the compiler. A step-by-step example will show you how to work with the debugger.

With this tutorial you will get an impression of the power of the StormC development system, and you will never want to 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 startup you will see a welcome message, which remains visible on the screen while components of StormC are loaded. After this you will see a horizontal toolbar near the top of your screen. It contains the most important

functions of StormC, close at hand for quick access.

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 a new project

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

What is a project?

A project is the summary of all related parts of your program, e.g. C, C++ and assembler sources, headers, object files, link libraries, documentation, graphics, pictures and other resources. Through the separation in various sections you will always keep an overview. The project manager 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 are checked by the project manager. So the project manager knows by itself, that a C source must be recompiled if a ".h" header which was included in the ".c" source has been altered.

At the click on the icons "Make" or "Run" all dependencies are examined. The Make routine now decides which module of the program must be compiled again. "Run" only differs from "Make" in that after 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. Select the directory "StormC:" and enter the path and file name "Hello World/Hello World". The suffix ".¶" is appended automatically and indicates that this is a StormC project. You can enter this extension manually by typing <ALT> + <P>.

You may wonder why an empty project should be stored, even when the project has no contents yet. We recommend doing this because now the paths to the sources and resources can be written relative to the project 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 editor 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 hide/show end-of-line marks
T = toggles tab marking on/off
S = toggles space marking on/off
I = toggles auto-indentation on/off
O = toggles overwrite mode on/off
R = toggles write protection on/off

The next field indicates whether the text has been modified.

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 manager looks at the file-name extensions to recognize the different file types.

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

Select the menu "Project" and the item "Select name of the
program". Make sure that the project window is the active window.

1.14 StormC.guide / ST_Compile

Compiling

Click on the project icon "Make". The compiler's error window is opened and the translation of the source starts. During translation some messages may be printed in the error window.

If an error occurs during compilation, a description will be printed here which contains the (possible) cause of the error and the number of the line in the source file where it occurred.

Double-clicking on an error message will get you back to the editor and to the source line that caused it. Make your corrections and compile again.

1.15 StormC.guide / ST_PRGStart

Starting a program

After successfully compiling and linking your program, 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 verifies that all modules have been compiled. If this is not the case, it now starts the compiler. After successful translation, the program is started automatically.

The RunShell

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

You will have noticed that when you start your program, 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 StormC's RunShell it is possible to debug the program after running it. If an error occurs 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 its "Resource Tracking" capability. To allow this all system functions pertaining to resource handling (AllocMem(), OpenWindow(), Open(), ...) are recorded. When the program is finished, the Resource Tracker checks if there are resources which have not been freed, or that have been freed more than once. You can now go directly to the source and make the necessary 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 also be set to any value between -128 and +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 allocated resources are released (storage and signals are released; screens, windows, requester and files are closed). As a result no "dead" tasks will remain in your system, which can normally emerge quite easily from inadvertently programmed endless loops. This feature will save you a lot of time, because it is a much faster way of cleaning up your Amiga than rebooting it.

In our example the program is processed so quickly that the RunShell window will be closed again almost immediately. The next example will show you more of the RunShell and how to 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 such as the "printf" function. To close it simply click on the window’s close gadget.

1.16 StormC.guide/ST_Debug

The debugger

The debugger is required for fast detection of any bugs in your program. It allows you 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 operation 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" are activated.

Start the compiler with a click on the debugger icon. The system now checks 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 started in debug mode.

Depending on the preset preferences the module window, the active-variables window and the monitored-variable window will be opened.

Furthermore the module which contains the main function is opened and its source indicated, starting with the main() function.

You will note that the contents of the editor window have changed a bit. The

first column of the text is placed further to the right, so that the freed column can be used to show breakpoints. The breakpoint fields are only in the lines at which the program can be stopped. If you click on one of these points they marked with 'X'. This indicates an active breakpoint.

Set a breakpoint directly after the "OpenScreen" call. If the "current variables" window is not active, open it 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 to "OpenScreen", the program is executed up to here. A new screen is opened and the program is halted.

The next function "GetRGB32" provides an array of unsigned long characters with data. We want to give this array a closer look.

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

Q = show corresponding place in the source code
F = show member function of a class type
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

First we need to put the array variable colortable into the inspect window. In the window "current variables", select the variable "colortable" with the mouse and click on the symbol "I" at the upper edge of the window.

Now execute three single steps and observe how the contents of the inspect window change. Click on the "go one step" symbol in the RunShell window twice. The function "GetRGB32" loads the values of the screen view structure into the array "colortable". Note how simple it is to monitor variables with the inspect window!

You may want set further breakpoints and play with the debugger and its functions. You will become familiar with the controls very quickly.

To exit the debugger, terminate the program started in the debugger mode. All debug windows will be closed automatically as well as the RunShell.

1.17 StormC.guide / ST_Sektion

Sections of a project

The files in a project are categorized into sections automatically by filename extension and, in the case of documents, by entire name as well. If you add a ".c" file to a new project, the project manager creates a new section called "sources" containing this file. When adding further sources to the project, the existing "sources" section is simply expanded.

```
The following sections are currently recognized:

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
  ".¶"

Amiga Guide
  ".guide"

Program
```

1.18 StormC.guide / ST_Owns

```
Peculiarities of StormC

Despite the standard ANSI C specifications each compiler has its own
```

peculiarities. These specialities are introduced with "#pragma". As with #include, #pragma lines are interpreted and executed by the preprocessor.

Modes of the compiler

#pragma -

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

#pragma +

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

Chip and Fast RAM

The architecture of the Amiga is a bit unorthodox in some respects; for instance there are different classes of RAM.

Normally the programmer is only interested in the answer to "Chip or Fast RAM?", because eg. graphical data needs to be allocated in Chip memory. StormC therefore offers the pragmas "chip" and "fast" .

All static data declared after the line

#pragma chip

are loaded into Chip RAM.

#pragma fast

switches into normal mode, in which your data is placed in whatever memory is available (preferably Fast RAM).

Breakpoint control

Breakpoint control has nothing to do with the Breakpoints in the debugger, but rather with the ability of the compiler to create programs which can be interrupted by the user. This is pleasant both during 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 feature works by testing the signal bit which is set when the user presses the keys <CTRL> + <C> or by the CLI command "break", at every location in the source file with a minimum of one time per loop. Needless to say, this costs 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, you can use the "break" pragma.

#pragma break +

Enables insertion of breakpoint code.

pragma break -

Disables insertion of breakpoint code.

If you press <CTRL> + <C> when the interrupt option is enabled, this corresponds to executing "exit(900)" from the program. All files are closed and all blocks of memory and other allocated resources are freed. Functions registered with "atexit()" are executed. As with a normal "exit()", no destructors are called.

OS calls

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

Such a declaration basically consists of four parts:
- name of the library-base variable.
- function offset as a positive integer.
- name of the function (must have already been declared); to avoid ambiguities these names may not be overloaded.
- list of parameters (represented by a corresponding list of register names between parentheses)

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 yourself since everything is included with the Amiga libraries.

Joining lines

Line breaks are of little consequence in C and C++, but they are significant to the preprocessor as each instruction must fit in exactly one line. Sooner or later you may come to the point, e.g. in an extensive macro definition, where you need to write a very long line. For this case there is the backslash. Any line ending in "\" is joined with the following line; both the backslash and the line break are completely ignored. Example:

```
# define SOMETHING \
47081115
```

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

Predefined symbols

The preprocessor knows many predefined macros. Some are defined by the ANSI C standard, others are part of C++ or particular peculiarities of StormC. These macros can not be redefined.

__COMPMODE__

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

__cplusplus

In StormC the macro "__STDC__" is defined in C as well as in C++ mode. If you want to check whether your source is being compiled in C++ mode, this must be done with the macro "__cplusplus" .

__DATE__

The macro __DATE__ is expanded to the date of the compilation. This is very useful if you want to furnish a program with a 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 in the standard "hours:minutes:seconds" format.

__FILE__

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

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

The value of the __FILE__ macro 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 it is not defined.

__STORM__

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
When selected from the icon or project window, a new project is opened; this corresponds to a click on the icon "new project".

If the active window is an editor window, a new editor window is opened; this is the same as clicking on the "new text" icon.

Open... A-O
If the icon or project window is active a project will be opened. When selected from the editor window a text file is loaded. In either case, a standard ASL file requester will pop up, asking for an input file. You can choose Open... from the icon bar as well.

Save A-S
If the project window is active, the project is saved. This corresponds to a click on the icon "Save Project".

If an editor window is active, its text is saved. This corresponds to a click on the icon "Save Text".

Save as...
The Save file requester is opened; here you can select a file name for your project or your text.

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

Save as project pattern
This menu item is only accessible from the project window. The project pattern is a file containing preset project preferences which is loaded whenever you set up a new project. You can set the default options for future projects here.

This includes 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 Manager.

Save all
With this menu item all source and projects which have not been written do disk yet will be saved. If no file names have been selected for some sources or projects the Save file requester is opened for each missing filename.

Add files...
This menu item is only accessible from a project window. A file can be selected from the file requester, which is then imported into the Project Manager. Depending on filename extensions, 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 editor window is active. With a click the file of the active editor window is placed in the corresponding project section. In contrast to "Add files" the file requester does not appear.

Choose program name...
Of course each program requires a name. As a program may consist of many modules, this can not be done automatically. With the help of the file requester you can enter the program name and choose an appropriate location on your disk.

Close A-K

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

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

About
This will show a requester in which you will find information about the product, copyright, our current telephone and fax number, and our email address.

Quit A-Q
Exit StormC. If any sources or projects have not been stored yet, you will see a message reminding you of this.

Edit

Mark A-B
This menu item is only available from editor windows. It switches the marking mode of the editor.

Cut A-X
This menu item is only available from editor windows. The marked text area is copied to the clipboard and deleted from the source text.

Copy A-C
This menu item is only available from editor windows. Like "Cut", this copies text into the clipboard; the difference is that with "Copy" the block is not deleted from your source text.

Paste A-V
This menu item is only available from editor windows. It inserts the contents of the clipboard into your text at the current cursor position.

Delete
When selected from an editor window, the marked area is erased from your text. The erased text is not copied into the clipboard, but if necessary you can still restore your text to its previous state using the "Undo" option.

When selected from a project window, the module marked in a section is removed from the project. "Undo" is not possible here!

Undo A-Z
This menu item is only available from editor windows. "Undo" reverts the most recently invoked editor function.

Redo A-R
This menu item is only available from editor windows. With "Redo" you can take back the most recent "Undo".

Find & Replace... A-F
Find & replace can of course only be selected from the active editor window. In the dialogue box that appears after selecting this option, you may enter the search key and any text you wish to replace its occurrences in the text with.

This function can be used for searching as well. Just omit the replacement string and click on the "Find" gadget. The search direction can also be selected.

Using the cycle gadget you can set the options in more detail, in particular you may choose to ignore letter case and accents.

With the three gadgets at the lower edge of the dialogue you may execute the search commands in different ways.

"Find" simply searches for the 'find' string
"Replace" replaces the 'find' string with the 'replace' string once
"Replace all" replaces all occurences of the 'find' string found with the 'replace' string

Find next A - " . "
This menu can only be selected from the editor window.

"Find next" repeats the most recent find command without opening the dialogue box again.

Replace next A - "-"
This menu can only be selected from the editor window.

"Replace next" repeats the last-made replace command without opening the dialogue box first.

Compile

Compile...
The menu item "compile. .. " is available if an entry is marked in the sources section of the project. It may be also be selected if the source indicated in the active editor window is found in the active project.

With this function you may compile individual modules.

Make... A-M
This menu item is available only from the project window or the error window. It may also be selected if the source indicated in the active editor window is found in the active project.

"Make" compiles all modules that have been altered since the last compilation, taking dependencies between program and header files into account.

Clicking on the "Make" gadget has the same effect.

Run...
This menu item is available from the project window or the error window only. It may also be selected if the source indicated in the active editor window is found in the active project.

If an up-to-date version of the program already exists, it will be executed. Otherwise a "Make" will be performed first, so that all changed modules are compiled. Afterwards the new program is executed.

Clicking on the "Run" gadget has the same effect.

Debug... A-D
This menu item is available only from the project window or the error window. It may also be selected 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 to the beginning of the first function (main) and execution of the program is halted at this position.

Clicking on the "Debug" gadget has the same effect.

Touch
This menu item is available from the project window only.

If you "Touch" an entry in the source section of the Project Manager it will be marked as "changed". Next time a "Make" is performed, this source is certain to be recompiled.

Touch all
This menu item is available from the project window only.

When selecting this item, all modules of the source section are marked "changed". Next time a "Make" is performed, all sources will be recompiled.

Save program as...
This menu item is only available from the project and error windows.

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

Windows

Error window...
This menu item is available from the project window only.

It opens the error window.

Modules...
This menu item is only available if the debugger is active. It opens the windows of the modules for which information can be printed. This normally includes all entries of the "sources" category in the project window for which a debug file has been generated. Instead of the source names you will find the file names of the object modules.

Current variables...
This menu item is only available if the debugger is active. It opens the window showing the current variables.

Monitored variables...
This menu item is only available if the debugger is active. It opens the window showing the monitored variables.

Options

Project...
These options are only available when the corresponding project or error window is active.

Include paths and pre-compiled headers

The preprocessor is a software module which processes the source files before the compiler gets to look at them. The preprocessor fetches definitions for e.g. the standard library functions from include files. Use the preprocessor 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" (ie. between double quotes), the preprocessor looks for them in the current working directory. If you enclose them between angled brackets <like this>, they are assumed to indicate names of standard include files and they are loaded from the appropriate directories. In the "Include Path" Listview you can choose one or more directories to search for the standard include files.

"Copy to:" gives you the option to speed up compilation by caching include files on the RAM DISK; but of course this will use up some more memory.

Another way to speed up compilation dramatically and even to save RAM is to use pre-compiled header files. To use this feature, tell the compiler where the header file ends and your program starts with "#pragma header". A simple way is to put the "#includes" of all header files that are not yours or are never changed (e.g. OS includes) right at the top of each module. Insert a line "#pragma header" directly below that. Below this point you may add your own header files and the rest of your program.

This pragma instruction has no effect unless you select the "Write header file" option and recompile the changed headers. As soon as the compiler reaches the pragma instruction the pre-compiled header files are written into the corresponding directory under the indicated name.

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 gadget just below the upper edge of the window.

Preprocessor

Here you can select what warnings the preprocessor should generate, and predefine any preprocessor symbols.

The preprocessor warnings are easy to configure, but the definitions need some explanation. Each entry you make in this Listview will have the same effect as writing it at the top of each source file with a "#define" in front of it. This lets you make global definitions such as "DEBUG" or define tokens such as "TRUE" and "FALSE".

Generating code and debugging

Here you may select the source translation mode (ANSI C or C++) and, in the case of C++, the processing of templates and the use of exception handling.

The next cycle gadget switches debug output generation on or off. If you want debug output the compiler generates additional files with the suffix ".debug" and ".link". These files are required by the debugger; they describe the

relation between sources and program code.

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

You can even produce assembler sources. The compiler creates additional ".o" and ".s" files. They contain assembler source interspersed with the corresponding C statements in your program.

If you enable "interrupt code generation" the compiler inserts a check for <CTRL> + <C> in every loop.

When creating code for the 68000, you should enable "32 bit multiplication"; this will cause library calls to be used for long word division and multiplication. This switch is ignored when generating code for higher processors.

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

The next cycle gadget selects the processor type. Please keep downward compatibility in mind: if you generate code for the 68060 the resulting program will not run on a normal Amiga 2000.

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

The last cycle gadget toggles between large and small data model.

Warnings

StormC distinguishes eight warnings which you can enable or disable according to your individual needs.

Linker settings

The path for link libraries may be set here; this is similar to the preprocessor’s include path.

The next cycle gadget 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. Such programs can not be run from CLI or Workbench.

You may also select whether linking should be done if an error occurs during compilation.

Running

If you run a program from StormC’s integrated environment you may specify command-line arguments and set the program’s stack size.

Load settings...
You can load a debugger settings file with the suffix "RUN". This item is only available if the debugger is active.

Save settings
You can store the complete debugger configuration (which windows are opened and their respective positions). The "StormSettings.Run" file is saved to the home StormC directory. This item is only available if the debugger is active.

Save settings as...
Stores the complete debugger configuration as above, but you are given the opportunity to choose a different name and location. This item is only available if the debugger is active.