

WERCS Language Details

This section details the language specific details of the name files produced by WERCS.

Assembly Language

If you have selected Assembler from the Language dialog box WERCS will produce a file with extension .I containing EQU statements of the form:

```
label EQU 1
```

If you have a saved a resource file called TEST.RSC you would then include the constants from the name file using:

```
INCLUDE TEST.I
```

The characters allowed in names are:

A-Z, Q-Z and _ as the first character and:

A-Z, Q-Z, 0-9, _ and . in subsequent characters.

Although designed with DevpacST in mind, the .I file can be used with other assemblers that follow the Motorola standard.

BASIC

If you have selected BASIC from the Language dialog box WERCS will produce a file with extension .BH containing CONST statements of the form:

```
CONST label%=1
```

If you have a saved a resource file called TEST.RSC you would then include the constants from the name file using:

```
rem $include test.bh
```

The characters allowed in names are:

A-Z, Q-Z as the first character and:

A-Z, Q-Z, 0-9, _ and . in subsequent characters.

The .BH file is designed with HiSoft BASIC and Power BASIC in mind, and adaptation to other BASICS for the Atari ST is straightforward straightforward so long as the BASIC is suitable for serious GEM work.

C

If you have selected C from the Language dialog box WERCS will produce a file with extension .H containing #define pre-processor statements of the form:

```
#define label 1
```

If you have a saved a resource file called TEST.RSC you would then include the constants from the name file using:

```
#include "TEST.H"
```

The characters allowed in names are:

A-Z, Q-Z and _ as the first character and:

A-Z, Q-Z, 0-9, and _ in subsequent characters.

All sixteen characters of the name are significant as is the case with Lattice C 5. This may cause problems with some other C implementations, such as the HiSoft C Interpreter, if your names have the first eight characters the same.

FORTRAN

If you have selected FORTRAN from the Language dialog box WERCS will produce a file with extension .INC containing PARAMETER definitions of the form:

```
INTEGER*4 LABEL  
PARAMETER (LABEL=1)
```

The characters allowed in names are:

A-Z, Q-Z as the first character and:

A-Z, Q-Z, 0-9, and _ in subsequent characters.

All sixteen characters of the name are significant as in Prospero FORTRAN; the WERCS output was designed for use with this compiler and with Prospero's GEM bindings; they may not be appropriate for other FORTRANs.

We would like to apologise for the lack of an example FORTRAN program; we do not have the necessary FORTRAN expertise to produce this.

Modula-2

If you have selected Modula from the Language dialog box WERCS will produce a file with extension .DEF containing a definition module. For example, if you have saved a resource file called TEST.RSC, the file will be of the form:

```
DEFINITION MODULE TEST;  
CONST label=1;  
.....  
END TEST.
```

If you are writing a one-module program you can use an implementation module of the same name.

Otherwise, you will need to write an implementation module like this:

```
IMPLEMENTATION MODULE TEST;  
END TEST.
```

To access the constants from your other modules use:

```
FROM TEST IMPORT label;
```

or

```
IMPORT TEST;
```

and then access the labels as, for example, TEST.label.

Remember not to use the same name as your main module if you are using this method.

The characters allowed in names are:

A-Z, 0-z, \$ and _ as the first character and:

A-Z, 0-z, 0-9, \$ and _ in subsequent characters.

All sixteen characters of the name are significant.

Although designed for use with FTL Modula-2, the name files may be used with any Modula-2 compiler that follows the Third Edition of Wirth's book.

Pascal

If you have selected Pascal from the Language dialog box WERCS will produce a file with extension .INC containing constant definitions of the form:

```
CONST  
label=1;  
.....
```

If you have a saved a resource file called TEST.RSC you would then include the constants from the name file using:

```
{ $I TEST.INC }
```

The characters allowed in names are:

A-Z, 0-z as the first character and:

A-Z, 0-z, 0-9, and _ in subsequent characters.

All sixteen characters of the name are significant.

Although our example programs are for Personal Pascal, the .INC files generated are suitable for use with most Pascal compilers.

The WTEST Example Programs

Compiling WTEST

To illustrate the most common resource-handling programming requirements, we supply an example program written in a variety of languages. The programs all do the same thing but the implementations vary according to the language used. A ready-to-run version called WTEST.PRG is supplied on the master disks; it needs WRSC.RSC to run.

To re-compile WTEST first you need to run WERCS and then use the LOAD command from the File menu to load WRSC.RSC from your backup disk. When WRSC.RSC is loaded the name file, WRSC.HRD containing the names of the forms and objects will be loaded as well.

All the following commands are on the File menu. Click on Language and then select the language that you wish to use by clicking on the appropriate radio button in the dialog box. Next save the file using Save; this will save the WRSC.RSC resource file, the name file WRSC.HRD and also a file for the language of your choice. Next use Quit to leave WERCS.

Pascal

If you have selected Pascal from the Language dialog box WERCS will produce a file with extension .INC containing constant definitions of the form:

```
CONST  
label=1;  
.....
```

If you have a saved a resource file called TEST.RSC you would then include the constants from the name file using:

```
{ $I TEST.INC }
```

The characters allowed in names are:

A-Z, 0-z as the first character and:

A-Z, 0-z, 0-9, and _ in subsequent characters.

All sixteen characters of the name are significant.

Although our example programs are for Personal Pascal, the .INC files generated are suitable for use with most Pascal compilers.

The WTEST Example Programs

Compiling WTEST

To illustrate the most common resource-handling programming requirements, we supply an example program written in a variety of languages. The programs all do the same thing but the implementations vary according to the language used. A ready-to-run version called WTEST.PRG is supplied on the master disks; it needs WRSC.RSC to run.

To re-compile WTEST first you need to run WERCS and then use the LOAD command from the File menu to load WRSC.RSC from your backup disk. When WRSC.RSC is loaded the name file, WRSC.HRD containing the names of the forms and objects will be loaded as well.

All the following commands are on the File menu. Click on Language and then select the language that you wish to use by clicking on the appropriate radio button in the dialog box. Next save the file using Save; this will save the WRSC.RSC resource file, the name file WRSC.HRD and also a file for the language of your choice. Next use Quit to leave WERCS.

One more general point: if you edit a resource file and change its structure you will need to re-compile your program, in case any constants have changed.

C

WTEST.C is the source file for use with Lattice C 5. This is an extended version of the standard WTEST, only supplied with Lattice C 5, which rather than using square buttons, uses rounded ones. Several other headers files supplied as part of Lattice C 5 are also used.

Assembly Language

WTEST.S is the source file for use with DevpacST2. As well as the WRSC.I file produced by WERCS you will need GEMMACRO.S and AESUB.S from your DevpacST master disk.

BASIC

WTEST.BAS is the HiSoft BASIC version which will also compile under Power BASIC. It needs the GEMAES.BH file from your BASIC master disk as well as the WRSC.BH file produced by WERCS.

Modula-2

WTEST.MOD is the FIL Modula-2 version. You will need to compile the file WRSC.DEF produced by WERCS before you compile WTEST.MOD. You will also need to compile WRSC.MOD from your WERCS backup disk before you link.

Pascal

WTEST.PAS is the Personal Pascal version; to compile this you will need GEMSUBS.PAS from your Personal Pascal backup disk.

WTEST structure

The different versions of WTEST all do the same thing but the implementations vary according to the language used. We have tried to keep variable names and procedure/function names as consistent as possible.

The program is deliberately over-simplified; it manages to avoid calling the VDI completely and gets away with an event_mesag, avoiding the dreaded event_multi. The general structure of the code in all the programs is as follows:

Procedure INITIALISE

This does the required GEM initialisation then loads the resource file. The tree address of the menu is found and the menu installed. The usable screen size is found and certain global variables initialised.

Procedure SETDESK

This sets the new desktop pattern to be a particular address and forces the AES to re-draw the whole screen.

Procedure DEINITIALISE

Resets any installed desktop, removes the menu bar, frees the resource, then does any required GEM de-initialisation.

Procedure HANDLE_DIALOG

A general dialog box handler which starts by centring and drawing the box. User interaction is handled by form_do and, on return, if the exit object was a Button, it is de-selected.

Procedure SET_TEDINFO

This allows a particular TEDINFO structure to have its data portion set to a particular string.

Procedure GET_TEDINFO

Allows a particular TEDINFO structure to return its data portion.

Procedure ROUND_BUTTON

This is the code that the ProgDef in the Lattice C 5 version of WTEST uses.

Procedure OBJ_INIT

This modifies the button objects in the dialog box to be ProgDets. The strings already there are saved and used subsequently for the round buttons. It also determines the AES's VDI handle for the use of ROUND_BUTTON. This routine is only supplied in the Lattice C 5 version.

Procedure SET_BUTTON

This allows one particular radio Button to be set from a group. If invalid parameters are specified the routine will never finish.

Procedure GET_BUTTON

Allows a group of radio Buttons to be interrogated to see which is selected.

Procedure TEST_DIALOG

This handles the particular dialog box in the resource file. It implements proper cancelling - that is, if it is cancelled, the Button state and Text entry are left alone.

Procedure HANDLE_MENU

This is the menu-click dispatcher; it takes various actions, depending on which menu item has been clicked, and also de-selects the menu title.

Procedure MAIN

This is the main loop, acting only on MN_SELECTED message events. In a proper program `evnt_multi` would be used and a far greater selection of cases would have to be dealt with.

HRD file format

.HRD files consist of a header record, any number of variable length data records and then an end-of-file record.

HRD Header record

| Name | Size | Meaning |
|------------|------|---|
| version | word | 1 at present |
| autonaming | byte | 1 if auto-naming selected 0 if no auto-naming |
| langflag | byte | 1 if C, 2 if Pascal, 4 if Modula-2, 8 if FORTRAN, 16 if assembler, 32 if BASIC |
| autosnap | byte | 0 if no character-snap 1 if half character-snap 2 if full character-snap |
| casing | byte | 0 if mixed 1 if upper 2 if lower |
| autosizing | byte | 1 if auto-sizing 0 if no auto-sizing |
| reserved | byte | not used at present |

HRD Data Record

| Name | Size | Meaning |
|-----------|--------|---|
| type | byte | 0 if Form, 1 if Menu, 2 if Alert, 3 if Free String, 4 if Free Image, 5 if object (rather than tree), 6 if end-of-file record, 7 if record names a prefix rather than a name. |
| reserved | byte | not used at present |
| treeindex | word | number of tree |
| objindex | word | if object then object number within tree |
| name | varies | Name terminated with a single null |

LNG file format

The WERC.S.LNG file is a text file containing the information that WERC.S uses to work out which name file to produce, what to output and what is a valid name. You can modify this if you wish; though be warned that it is easy to produce files that your compiler won't like.

The file consists of a number of Language specifications followed by an end record. Each Language specification starts with a line like:

*LANGUAGE C

where C is the name of the language. This is used purely for documentation purposes; it won't affect the Language Dialog Box for example. The records are for C, Pascal, Modula-2, FORTRAN, assembler, and BASIC in that order.

The end record is a single line:

*END

and should be the last line in the file; the information following it will be ignored.

The other lines in the language specification may appear in any order and can be one of:

*SOURCE .H

This specifies the extension of the source file that WERC.S will generate (.H in this example); the full stop (.) must be present.

*SIGNIFICANCE 16

Specifies the number of significant characters in names; minimum 1 maximum 16. The default is 16.

*INITIAL a-z,A-Z,_

Specifies the characters that are allowed as the first letter of the name. The hyphens (-) indicate a range of letters. The commas may be omitted. The equivalent of this would be

*INITIAL a-zA-Z_

which is far less obvious. Do not put spaces in the INITIAL string. The default is Q-Z, A-Z.

*FOLLOW a-zA-Z,0-9,-

specifies which characters are allowed in names other than in the first position. The syntax to specify the characters is the same as for *INITIAL. Default is Q-Z, A-Z, 0-9.

*INIT

*TREE

*OBJECT

*EXIT

These commands specify the text that is generated in the source file. The *INIT text is generated once at the top of the file, *END at the end of the file, *TREE for each named tree in the file and *OBJECT for each named object. Each entry may be more than one line long, the entry being terminated by the next * command. The following special pairs of characters may be used:

| Character | Meaning |
|-----------|--|
| % F | base file name (without drive, directory or extension), |
| % T | name of the current tree, |
| % N | name of the current object, |
| % V | the value of the current object (for *OBJECT) or tree (for *TREE), |
| % % | a single % character. |

Of the above only %F and %% should be used in *INIT and *EXIT. The default for *INIT, *EXIT, *OBJECT and *TREE is no text at all.

See the WERCS.LNG file for an example. If you want to generate code for a language that is not supported already, modify the definition of another language that you do not use. If you are adventurous you can even change the name of the Button in the Language box in WERCS.RSC but if something goes wrong don't blame us!

Appendix C Converting to Lattice C 5

Lattice 3.04

Old Lattice 3.04 programs are probably the easiest programs to convert to Lattice C 5. Often few or no changes will be required. There have however been many changes to gain ANSI conformance on the runtime libraries and some rationalisation of names to match those used by Lattice C on other architectures. The important changes for upgrade users are:

- cbs has been changed so that it only deals with the type `Int`.
- `mAlloc`, `cAlloc`, `free`, `reAlloc` have been extended to respond correctly to passing of ANSI NULL arguments. Note that this means that *any* use of `mAlloc(0)` will return a NULL pointer. Also note that `free` returns the 'type' `vold`, not `Int` as with the previous release.
- An extra level of data hiding has been introduced between the ANSI file and GEMDOS file handling. In particular `fileno` no longer returns a GEMDOS file handle, but instead an *internal* handle. The `_chkufb` function may be used to obtain the mapping from internal to GEMDOS handles.
- `dcreat`, `dclose`, `dopen`, `dread`, `dwrite` and `dseek` have had an underscore prefix added. Note also that the level of functionality at this level is significantly greater than before, as many of the GEMDOS anomalies are removed at this level.
- ^Z is no longer recognised as end of file marker in text files; this usage was not widespread and invariably caused confusion.
- `isdata`, `isdir`, `isstatic`, `isauto`, `isheap` and `ispptr` were hang-overs from the MS-DOS implementation in version 3 and not strictly relevant to the 68000 environment and so have been removed.
- Almost all internal variables have been renamed or removed, any references to such variables should (in general) be removed.
- The values placed in `_OSERR` are now the positive GEMDOS error codes. This change was to increase conformity with other Lattice compilers.

- `cllmem` and `blldmem` are no longer relevant to the new dynamic memory manager (cf. the old static memory manager); any calls to these functions are best removed.
- The `exec` family of functions are no longer available, they did not perform as described under the old documentation anyway (in fact they were fork synonyms). `TOS` is incapable of performing exactly the operations required to implement these functions as previously described.
- The `-n` flag on the compiler has the opposite sense to that under 3.04, i.e. `-n truncates` to eight characters rather than increasing the significance to 32 characters.
- The Line-A functions have *all* been implemented, and a correct header file made available together with many extensions. In particular, the `showmouse` macro from `linea.h` has been corrected to include the `hide` depth parameter; refer to the Line-A documentation for details.
- In conformance with ANSI, `memcpy` is not guaranteed to perform correctly when blocks overlap. Under version 3 this case was handled correctly.
- `_MNEED` is defunct due to the dynamic memory model used. It may be used to specify an initial heap size (e.g. if you have a program which will have only very small `malloc` requirements) although it no longer specifies the maximum heap size. In particular negative values are ignored.
- The GEM AES and GEM VDI header files have been split into two, although both may be included via `#include <gemlib.h>`.
- Many of the types in the AES and VDI prototypes have been changed to reflect more natural type usage. This should not, however, affect the code generated.
- The AES, VDI and Line-A libraries were re-written from scratch, hence any bugs in the old libraries are most unlikely to exist in the new libraries.

Other Differences

- This section lists other differences from version 3; hopefully all important conversion differences are listed above. Here is the list of the improvements and changes in Version 5, relative to Version 3:
- The compiler now uses sequence points to ensure correct evaluation and side effect generation according to the standard.

- The compiler now includes a full ANSI pre-processor with string facilities, token facilities and appropriate scoping of substitution symbols. The `defined()` directive is also supported. In addition, `__DATE__` and `__TIME__` provide the date and time of compilation.
- The `CONST` and `volatile` keywords are supported.
- Function prototypes may now include an optional parameter name. Also functions taking a variable number of arguments may be indicated with ellipses '...'.
- String literals may now be concatenated to allow easier coding of long strings.
- The cast operation (`VOID *`) correctly coerces a type without any warning.
- Many diagnostic messages have been added to detect programs that do not conform to the ANSI standard.
- The compiler now recognises several new keywords:

| | |
|--------------------------|---|
| <code>signed</code> | Overrides any default unsigned options. |
| <code>near</code> | Declares a data item to be addressed relative to the global base register. When used with a subroutine, it indicates a PC-relative subroutine call. |
| <code>far</code> | Declares an item that must be addressed with a full 32 bit address. |
| <code>huge</code> | Same as <code>far</code> . |
| <code>__regargs</code> | Defines a subroutine that is to be called with register parameters. |
| <code>__stdargs</code> | Defines a subroutine that is to be called with standard stack parameters. |
| <code>__asm</code> | Defines a subroutine that takes its parameters in a specific register |
| <code>__saveds</code> | Defines a subroutine that is to load up the global base pointer upon entry. |
| <code>__interrupt</code> | Defines a subroutine that may be called from interrupt code. |

- To provide faster compilation of a large project, the symbol table may be saved out to disk so that it can be used in future compilations. If you run a large header file through the compiler and save it in this way, subsequent compilations using that header file will be much faster.
- The compiler provides an option to ignore redundant `#include` statements (i.e. several `#include` statements that refer to the same file).
- The search rules for `#include` files have been modified to conform to standard UNIX search conventions.
- It is now possible to disable particular error messages as well as to change the severity of most messages. Along with this, it is now possible to specify a maximum number of errors allowed in a compilation so that the compiler will abort.
- We have implemented an improved form of error recovery for many of the common mistakes to eliminate many of the situations that resulted in the cascade of errors.
- All of the compiler messages have been moved to a separate file to simplify adaptation of the compiler to non English language environments.
- In order to produce a compiler that fits well on a smaller system, we have elected to provide a big version of the compiler that includes some additional features. If you wish to take advantage of these features (at a cost of about 15K), you must use this big compiler instead of the standard one.
- The big compiler provides a full listing ability including macro expansion display, nest level counting, and include file listing. This listing may also include an optional cross reference of all variables, `#define` values and structure tags.
- The big version of the compiler may be used to generate prototype files of all functions encountered in a module. This eliminates the potentially tedious task of constructing the list of prototypes for all functions in a project.
- The compiler generates instructions optimised for each of the 680x0 family processors including support for the address modes found on the 68020 and 68030.
- The compiler provides an option to generate in-line floating point instructions which directly access the 68881 and 68882 maths co-processors. This code takes advantage of register tracking.
- You may now instruct the compiler to choose code sequences optimised for space or for time.

- In addition to the `-r0/-r1` flags, you may freely mix the style of subroutine calls with the `near` and `far` keywords. Those declared `near` will be referenced with the `pc`-relative addressing while `far` will use the full 32-bit addressing.
- Data may be addressed much more freely with the `near` and `far` keywords. These control the type of addressing to be used for external data. Only those items declared `near` will be addressed as a 16-bit offset from the A4 register. All others will be accessed with a full 32-bit address.
- When the `-Cs` option is used, string constants will be placed in the code section. This option is beneficial when using the `-b0` option.
- Two styles of register parameters are supported. The `-rr` option causes the compiler to place, automatically, up to four parameters in registers for subroutine calls. The `__osm` keyword may be used in conjunction with a register specification list to cause the compiler to pass parameters in a given register.
- The compiler now tracks the condition codes affected by the generated code in an attempt to avoid generating unnecessary test instructions.
- Stack cleanup on subroutine calls is delayed as long as possible to allow coalescing and even elimination of the cleanup across multiple calls.
- The bitwise Boolean operations generate better code for dealing with constant values.
- Division by 2, 4, or 8 no longer generates a subroutine call. The compiler generates inline code to normalise and perform the calculation.
- Bit shift operations have been re-written completely. The compiler now generates optimal shift sequences for all constant values.
- The compiler attempts to place as many variables as possible in registers unless this feature is explicitly disabled.
- The code generator now takes full advantage of *all* 68000 address modes including auto increment and `PC`-relative indexed. Tracking of indexing operations allows the compiler to suppress unnecessary additions and substitute indexed address modes.
- Loading of specific constants has been optimised to generate the optimal code sequence and avoid `MOVE.L #` as much as possible.

- Several new built-in functions have been added:

| | |
|--------|---|
| abs | Return the absolute value of an integer. |
| __emit | Insert a hex word into the instruction stream. |
| ...fpc | Generate MC68881 transcendental operation. |
| geta4 | Force loading of the global data register. |
| getreg | Obtain the contents of a specific 68000 register. |
| max | Return the larger of two integers. |
| memcmp | Compare memory blocks. |
| memcpy | Copy memory block. |
| memset | Initialise memory block. |
| min | Return the smaller of two integers. |
| putreg | Directly store into a specific 68000 register. |
| strcmp | Compare strings. |
| strcpy | Copy strings. |
| strlen | Obtain string length. |

- Many small code optimisations suggested by users have been implemented. In particular, the compiler no longer will generate NOP instructions. Also, MOVEM instructions in the prologue/epilogue that reference a single register are converted to MOVE instructions.
- All GEMDOS/BIOS/XBIOS functions are available via inline TRAPs eliminating the overhead of 'stub' based methods.
- switch statements on values which are in the range of a short are converted to use the more efficient code.
- The entire run-time library has been re-written to take advantage of better algorithms. The most important routines were recoded in assembly language.

HiSoft C

Converting HiSoft C programs to Lattice C 5 is normally fairly simple, requiring few changes to the source code. The following options are recommended to ensure compatibility with the HiSoft C model:

- OJ Force all characters to unsigned; HiSoft C always considers plain characters to be unsigned. The use of this option will ensure that any dependencies on this behaviour will not cause unexpected effects.
- fd Force all floating point declarations to be of type double as HiSoft C only supports the double real format.

The following points should also be taken into account when converting code:

- If you are using the HiSoft C toolbox the main function should be renamed as lc_main, so that the special initialisation code runs before your program.
- The HiSoft C toolbox is supplied already compiled as a library for use with Lattice C 5 and can simply be linked in - see the **Installation Guide**.
- HiSoft C programs which do not use the GEM toolbox, but do use GEM, must use appl_init and appl_exit. Their use is optional under HiSoft C whereas Lattice C 5 uses these calls to perform some initialisation itself. Failure to call these will result in mysterious crashes.
- The HiSoft C functions strgetfn and strspfn were re-named from their traditional Lattice names. The functions strmfnc and strsfnc respectively should be used in their place. The easiest solution is to define this mapping on the command line to Lattice C (since HiSoft C would not allow the mapping within the program) viz:

```
-dstrgetfn=strmfnc -dstrspfn=strsfnc
```

Where code cannot be made to run successfully under both systems, the IC and LATTICE pre-processor symbols can be used to distinguish between the translation environment; HiSoft C always defines IC, Lattice C 5 always defines LATTICE. Hence one can write:

```
#ifdef IC
/* HiSoft C specific code */
#else
/* Lattice C version of the code */
#endif
```

Appendix D GST Support

LinkST, The GST format linker

Introduction

LinkST is a linker that links GST-format files. In general we recommend that you use the Lattice C Linker, CLink rather than LinkST since CLink is faster and has more facilities. LinkST does have one advantage over CLink though: it will let you link with other languages that produce GST format files.

For example, if you have some assembly language written with GenST, the assembler supplied as part of HiSoft DevpccST2, then you could use GenST to produce a GST-format file and then link it, together with your C code and the GST libraries, to produce an executable program. If the assembly language module you have is short or doesn't use the extensions to the Motorola standard provided by DevpccST, we recommend that you convert your assembly language to the Lattice assembler, *asm*, so that you *can* use CLink.

If you wish to link your Lattice C code with code written in another high level language that supports the GST format (producing a so-called *mixed language* program) or you wish to use a third party library provided in GST format, then you will need to use LinkST.

Producing mixed language programs is not for the faint hearted; you need to ensure that the required start up code is provided for both languages and that the memory models, including register conventions, are compatible. Fortunately Lattice C 5 provides a wealth of options, so that you can match most popular conventions. See the section on **LC, The Compiler**.

If you are 'only' linking with assembly language, things are much simpler since you can modify the code that is being called to handle whichever memory model you wish to use.

Note that Lattice C 5 does not produce the other commonly used format on the ST, DRI linkable code, because this linker format is not sufficiently rich to support some of code constructs that Lattice C 5 generates.

Note

LinkST will only link GST-format files.

Compiling code in GST format

To produce code that can be linked with LinkST you will need to use the LC.TTP driver's -z option or, alternatively, convert a Lattice format file to GST format by using lc2gst as described at the end of this appendix.

If you are producing a program that is larger than about 32K using GST format then you cannot use the compiler's default small code model because GST linkers cannot generate ALVs, the clever method that CLink uses to allow you to use the small code model with large programs. Thus you will need to use the -f0 option when compiling GST-format libraries.

The GST libraries that we supply follow the same naming conventions as those in Lattice format only with the GST .bin extension. If you are using -f0, you'll need to use lcnb.bin, lcgmb.bin and lcrmb.bin as your libraries, which give you large code and large data. Thus to link with these you will also need to use the -b0 compiler flag.

There are a few other points regarding the use of GST format:

- The maximum amount of near data is restricted to 32K rather than 64K as with CLink.
- Lattice line number debugging information cannot be included in your executable program.
- There are some errors that CLink can spot that LinkST cannot; these are mainly mis-uses of memory models.

Invoking LinkST

As with most of the tools supplied with Lattice C 5 you can either run LinkST from the Desktop or from a shell and can either supply a complete command line or specify a link file which contains the required information.

The command line should be of the form:

```
<filename> <-options> [filename] [-options]
```

Options are denoted by a - sign then an alphabetic character, supported options being:

- B generate a true BSS section for any such named sections
- D debug - include all symbols in the binary file using DR standard 8 character format (for MonST2C or other debuggers)

-F force pass 2 of the linker, useful if you want to see all errors (as any pass 1 errors will, by default, stop the link before the second pass)

-L specify that all following filenames are library filenames

-M dump a map file showing the order of the sections and labels. The map filename will be the main filename with an extension of .MAP

-O specify object code filename, may be followed by white space before filename

-Q 'quiet' mode, which disables the pause after the link

-S dump a symbol table listing, The symbol table filename will be the main filename with an extension of .SYM

-T truncate to eight characters

-U force upper case

-W specify control file filename, defaults to .LNK extension

-X extended debug, using the HiSoft Extended Debug format for use with MonST2C.

Normally any file specified given are assumed to be input files, defaulting to the extension of .BIN, though if a .LNK extension is specified it will be taken to be a control file. After a -L option, filenames are all assumed to be library files.

The output file can be specified with the -O option on the command line, or using the OUTPUT directive in the control file. If there is more than one of these directives or options, the last one is used. If there is none given, then the first input filename specified in the command line or control file is used, with an extension of .PRG.

Example Command Lines

```
PART1 PART2 -d
```

Reads PART1.BIN and PART2.BIN as input files, and generates PART1.PRG as an output file complete with debugging information.

```
PART1 PART2 -o TEST.PRG
```

Reads PART1.BIN and PART2.BIN as input files, and generates TEST.PRG as an output file.

-o TEST.TOS START -l MYLIB -s

Reads START.BIN as an input file, selectively reads MYLIB.BIN as a library, and generates the output file TEST.IOS and the symbol listing file TEST.SYM.

LinkST Running

LinkST has two passes - during pass 1 it builds up a symbol table of all sections and modules, and during pass 2 it actually creates the output file. When it starts it prints a logon message, then reports on which files it is reading or scanning during both passes. This gives you some idea of what takes time to do, as well as exactly where errors have occurred.

If there is enough free memory at the end of pass 1, LinkST will use a cache to store the output file, which speeds up the process greatly. If it uses the cache it will write to the disk at the end of pass 2, and report the number of errors.

When the link finishes you will be prompted to press a key before quitting. This is to give you an opportunity to read any warning or error messages before returning to the Desktop. You can disable this pause by using the -q option, useful if you are using a CLI.

Error and warning messages are directed to the screen - if you want to pause output you can press Ctrl-S, while Ctrl-Q will resume. Pressing Ctrl-C will abort the linker immediately.

You can re-direct screen output to a disk file by starting the command line with

>FILENAME.TXT

or you can re-direct it to a printer by starting the command line with

>PRN:

to the parallel port, or

>AUX:

to the serial port.

If you do re-direct output in this way you should use the -q option as you won't be able to see the prompt at the end of the linking.

Control Files

The alternative way to run the linker is to have a control file for the programs which you are linking together.

If you require a lot of options which won't fit on the command line or you get bored of typing them, you can use a control file, which is a text file containing commands and filenames for the linker. The default extension is .LNK and the control filename is specified on the command line using the -w (for With) option. Each line can be one of the following:

INPUT <filename>

This specifies a filename to be read as an input file. The default extension is .BIN if none is given.

OUTPUT <filename>

This specifies the filename to be used for the output file. There is no default extension - you must specify it explicitly.

LIBRARY <filename>

This specifies a filename to be scanned as a library. The default extension is .BIN if none is given.

SECTION <sectionname>

This allows specific section ordering to be forced.

DEBUG

All symbol names included in the link are put in the output file so that debugging programs such as MonST2C can use them when the program is running.

XDEBUG

Similar to the debug option but uses HiSoft Extended Debug format for up to 22 character significance.

DATA size[K]

The BSS segment size is set accordingly. The size can be given either as a number of bytes or as a number of K-bytes (units of 1024). This option is particularly useful for the Prospero compilers which effectively use the BSS segment for their stack.

BSS <sectionname>

Specifies that the named section should lie in the GEMDOS BSS section area. This can save valuable disk space, but will generate errors if the section contains any non-zero data. This should not be used at the same time as the DATA statement.

TRUNCATE

Causes all symbols to be truncated to 8 characters. This is sometimes required to link assembly language with long labels to high-level language code with short labels.

UPPER

Forces all symbols to be automatically upper-cased. This is sometimes required when a compiler or assembler generates case-insensitive code.

Blank lines in the control file are ignored, and comments can be included by making the first character in the line a *, a ; or a !.

With the INPUT or OUTPUT directive, if the filename is specified as * it is substituted the first filename on the command line. This can be useful for having a generic control file for linking C programs for a particular memory model.

An example control file is:

```
* control file for linking large model gem programs
INPUT CNB
INPUT *
XDEBUG
LIBRARY LCGNB
LIBRARY LCNB
SECTION TEXT
SECTION DATA
BSS UDATA
```

Assuming this control file is called CPROG.LNK, the LinkST command line

```
TEST -w CPROG
```

will read as input files CNB.BIN and TEST.BIN, and scan the libraries LCGNB.BIN and LCNB.BIN. The object code, including extended debug information, will be written to TEST.PRG, as no output file was explicitly specified.

The two SECTION directives, above, ensure that the TEXT and DATA sections appear in the correct order in the output file. The BSS directive ensures that the UDATA section is treated as a true BSS section.

If you do not specify a drive name in the control file or on the command line, the default drive will be assumed. If you run LinkST from the Desktop, the default drive will always be the same as that containing the file on which you double-clicked; though if you run it from a CLI or from the editor this will not necessarily be so.

LinkST Warnings

Warnings are messages indicating that something might be wrong, but probably nothing too serious.

duplicate definition of value for symbol <x>

The symbol was defined twice. This can happen if you replace a subroutine in a module with one of your own, for example. The linker will use the first definition it comes across, and give this warning on the second.

module name is too long

Module names can only be 80 characters long.

comment is too long

Comment directives are only allowed to be 80 characters long (don't ask us why, we don't know!).

absolute sections overlap

Two absolute sections clash with each other.

SECTION <x> is neither COMMON nor SECTION

A section name was specified without defining its type.

LinkST General Errors

unresolved symbol <x> in file <x>

The symbol was referred to but not defined in the file. There may also be other files which refer to the symbol, but this error gives you a start in your search!

XREF value truncated

A value was too large to fit into the space allocated for it, for example a BSR to an external may be out of range.

bad control line <x>

An illegal line was found in a control file.

non-zero data in BSS section

A section wanted as a true BSS section contained non-zero data.

LinkST Input/Output (I/O) Errors

file <x> not found

Can't open output file <x>

Can't open map file <x>

Can't open symbol file <x>

Can't open input file <x>

I/O error on input file

disk write failed

filename <x> was too long

LinkST Binary File Errors

These are errors in the internal syntax of the input file, and should not occur. If they do it probably means the compiler, assembler or converter produced incorrect code.

missing SOURCE directive

Can occur if a file is not in GST format, for example a DRI file.

runtime relocation is only available for LONGs

attempt to redefine id of symbol <x>.

attempt to DEFINE <x> with <id> of zero

bad operator code 0x99 in XREF directive

bad truncation rule in XREF

wrongly placed SOURCE directive

bad directive <99>

<id> <99>not DEFINED as a SECTION but used as one

attempted re-use of <id> <99>as SECTION id

attempted re-use of <x> as SECTION name

section is COMMON but being used as though it's not

SECTION is being misused as COMMON

unexpected end of input file

'Linker Bug' Messages

These can be produced as a result of internal checks by the linker. If you get one please send us copies of the files you are trying to link!

GSTlib, The GST format librarian

GSTlib is a librarian that is designed for maintaining GST format libraries. When specifying filenames to GSTlib you must explicitly include the file extension, normally .bin.

GSTlib has a number of different possible command lines as shown below:

Replace modules

`gstlib r[vsq][a|b obmod] library [files...]`

This will replace any current occurrences of the modules contained in the given files. If a module is not already present in the library then it will be added. If the library does not exist then a new library will be created that just contains these files. The following additional options may be used

- v** (verbose). Echo when adding or replacing a module or creating a library.
- a mod** (after). If adding a new module insert it after mod.
- b mod** (before). If adding a new module insert it before mod.
- s** (sort). Sort the library so that it can be scanned by a linker in a single pass, if possible. If this fails due to circular references then the new library will be saved in libname.tmp and the original library left as it was.
- q** (quit). Wait for Return to be pressed before exiting GSTlib, for use with the Desktop.

Update modules

`gstlib u[vsq][a|b obmod] library [files...]`

This works in precisely the same way as the replace modules option except that the modules are only updated if the versions in the files are more up to date than the version in the library. Exactly the same options may be used as with the r option.

Load modules

`gstlib l[vq] prog.lib [files...]`

This option can be used to produce a library that contains just the modules that would be included when linking a program. The v and q modifiers have their usual meaning. For example:

`gstlib l prog.bin c.bin yourprog.bin lc.bin`

could be used to create a library containing the modules required by yourprog. You could then run LinkST without the need to scan the library. Once you have created a library using this option it is unwise to sort it subsequently.

Delete modules

`gstlib d[vsq] library [modules...]`

Deletes the modules with the given names. As usual v will cause the librarian to inform you of its progress, q will pause before exiting and s will attempt to sort the library after performing the deletions.

Move modules

`gstlib m[vsq][a|b obmod] Library [modules...]`

Moves the given modules to the end of the library unless either o or b are specified, in which case:

- a mod** moves the modules immediately after obmod and
- b omod** moves the modules immediately before obmod.

The other modifiers have their usual meanings.

Tabulate modules

`gstlib t[vvvvq] library [modules...]`

This form of command line displays information about the given modules in the library. If no module list is included, the information is given about all modules. The different levels of information are as follows:

- v** module names only
- vv** as per v and the size and date
- vvv** as per vv and the list of exported (xdef) symbols

vvv as per **vvv** and the list of imported (xref) symbols
vvvv as per **vvv** and a cross reference of the symbols
As ever, **Q** may be included to pause before **GSTlib** terminates.

Extract modules

```
gstlib x[vkq] library [modules....]
```

This extracts the given modules from the library. If no modules are specified then all the modules are extracted. Note that module names may have **.O**, **.C** or **.bin** appended to them depending on the tool that produced them. The additional modifiers are:

- k** keep date stamp from library on the extracted files
- v** inform the user of progress
- q** pause before exiting

If a module that is being extracted does not have a name it will be placed in a file called **dummy###.BIN** where **###** is a unique decimal number.

Librarian command files

If the command line to **GSTlib** contains an **@** sign, the name following this is taken as a command file name and the command read from this. Additionally such files may contain lines starting with **#**; these are treated as comments. Long lines may be split by using **** as the last character of the line.

Example command lines

```
gstlib tv lc.bin
```

List all the modules in the library **lc.bin**; there are quite a few!

```
gstlib xk lc.bin printf.o
```

Extract the module **printf.o** into a file called **printf.o** retaining the date stamp that it had in the library.

```
gstlib d lc.bin printf.bin
```

Remove the module **printf.bin** from **lc.bin**.

The source code to **GSTlib** is supplied; see the **Installation Guide** for details.

lc2gst, The Object File Converter

lc2gst is a tool for converting Lattice C object files to the GST format. It has three different forms of command line as follows:

```
lc2gst file.o  
lc2gst -o dir\ file.o  
lc2gst -o john file.o
```

The first form simply converts **file.o** to **file.bin**. Note that the **.o** is required.

The second form places the converted file in the directory **dir**; note that the terminating **** must be present. This option can be used to keep your GST format files in one directory.

The final form causes **file.o** to be converted to **john.bin**. Note that the **.bin** extension is always used, regardless of any explicit extension after the **-o**.

lc2gst can convert more than one file at once: simply include the other files to convert at the end of the command line. It does not, however, expand wildcards itself, so that you will need a shell that supports this, such as **Craft**, to provide this facility.

Appendix E

Quick Options Reference

| LC | LC1 | LC2 | |
|-----|-----|-----|--|
| -b | -b | | Base relative data |
| -b0 | -b0 | | Non-base relative data |
| -b1 | -b1 | | Base relative data |
| -B | | | Always use 'big' compiler |
| -c+ | -c+ | | Suppress structure messages |
| -ca | -ca | | ANSI compatibility |
| -cc | -cc | | Allow nested comments |
| -cd | -cd | | Allow \$ in identifiers |
| -ce | -ce | | Suppress error line printing |
| -cf | -cf | | Require function prototypes |
| -cg | -cg | | Process ANSI trigraphs (not implemented) |
| -c1 | -c1 | | Suppress multiple includes of same file |
| -ck | -ck | | Allow new keywords |
| -c1 | -c1 | | Forces long alignment of all external data |
| -cm | -cm | | Allow multiple character constants |
| -co | -co | | Enable old style preprocessor |
| -cq | -cq | | Strengthen aggregate equivalence type checking |
| -cr | -cr | | Allow register keywords |
| -cs | -cs | | Create only one copy of identical strings |
| -ct | -ct | | Enable warnings for tags used without definition |
| -cu | -cu | | Force all char declarations as unsigned char |
| -cw | -cw | | Shut off warning for return without a return value |
| -cx | -cx | | Treat all global declarations as externals |
| -C | | | Continue on error |
| -d | -d | | Enable debugging |
| -d0 | -d0 | | Disable debugging |

| | | |
|--------|--------|--|
| -d1 | -d1 | Enable debugging - dump line table |
| -d2 | -d2 | Generate symbol information |
| -d3 | -d3 | Generate symbol information, dump at every line |
| -d4 | -d4 | Generate full symbol information |
| -d5 | -d5 | Generate full symbol information, dump at every line |
| -dx=y | -dx=y | Define preprocessor symbol |
| -e | -e | Recognise extended character set |
| -e0 | -e0 | Japanese character set |
| -e1 | -e1 | Chinese character set |
| -e2 | -e2 | Korean character set |
| -Ex=y | -Ex=y | Define environment variable x with value y |
| -f | -f | Use standard Lattice libraries |
| -f8 | -f8 | Generate code for Motorola 68881 |
| -fa | -fa | Auto-detecting I/O based 68881 |
| -f1 | -f1 | I/O based 68881 maths |
| -f1 | -f1 | Use standard Lattice libraries |
| -fd | -fd | Treat all declarations as double precision |
| -fm | -fm | Use float as single precision and double as double precision |
| -fs | -fs | Treat all declarations as single precision |
| -gd | -gd | Cross reference defined symbols |
| -gc | -gc | Cross reference compiler provided files |
| -ge | -ge | List excluded lines |
| -gh | -gh | List header files |
| -g1 | -g1 | List included files |
| -gm | -gm | List macro expansions |
| -gn | -gn | Print narrow lines |
| -gs | -gs | List source |
| -gx | -gx | Produce cross reference listing |
| -Hxxx | -Hxxx | Read in header file xxx |
| -ix | -ix | Specify include directory |
| -j<n> | -j<n> | Disable message n |
| -j<n>e | -j<n>e | Make message n an error instead of a warning |

| | | |
|--------|--------|--|
| -j<n>1 | -j<n>1 | Disable message n |
| -j<n>w | -j<n>w | Enable message n as a warning |
| -L+ | | Specify additional linker objects |
| -La | | XADDSYM option |
| -Lb | | BATCH option |
| -Lf | | MAP option |
| -Lg | | GEM library |
| -Lh | | Hunk map option |
| -L1 | | Library map option |
| -Lm | | Lattice maths library |
| -Ln | | NODEBUG option |
| -Lq | | QUIET option |
| -Ls | | Symbol map option |
| -Lv | | Verbose option |
| -Lx | | XREF map option |
| -1 | -1 | Align objects on longword boundaries |
| -M | | Only compile modified source files |
| -m | -m | Generate code for Motorola 68000 |
| -m0 | -m0 | Generate code for Motorola 68000 |
| -m1 | -m1 | Generate code for Motorola 68010 |
| -m2 | -m2 | Generate code for Motorola 68020 |
| -m3 | -m3 | Generate code for Motorola 68030 |
| -ma | -ma | Generate code for all Motorola processors |
| -mc | -mc | Disable cleanup overhead reduction enhancement |
| -mr | -mr | Disable automatic registerisation |
| -ms | -ms | Generate code optimised for space |
| -mt | -mt | Generate code optimised for time |
| -n | -n | Retain only 8 characters for identifiers |
| -ox | -ox | Place object file in location x |
| -p | -p | Preprocess only |
| -pe | -pe | Generate prototypes only for externs |
| -ph | -ph | Generated precompiled header file |

Appendix F

The Lattice C Start-Up

Introduction

The Lattice C compiler is supplied with a wide range of differing startup stubs which perform various levels of initialisation and have differing impacts on the programs which may be written.

The stubs

4 different stubs are provided for the following purposes:

Standard - suffix none

This family of stubs provide the normal entry and exit code required by a GEM or TOS program designed to be run either from the Desktop or from another program via GEMDOS PEXEC. They perform full command line parsing via either the Atari extended command line method, or if this is not available via the command line embedded in the programs basepage. The memory available to the program is assessed and various internal variables are initialised to support the dynamic memory allocator (note that this allocator is careful to ensure that bugs in TOS pre-1.4 are not aggravated). The environment variables available are also parsed into the standard UNIX environ format.

The final call, from the startup code, is to the pre-main routine `_main` which initialises the standard I/O library prior to calling the `main` routine. If your program uses no standard I/O you may wish to declare your main function as `_main` to ensure that no superfluous library routines are drawn in.

Desk Accessory - suffix 'acc'

The desk accessory family of stubs perform the minimal initialisation required by desk accessories. The only 'normal' operations performed are the initialisation for the `CLOCK` function and the reading of the OS information. The stack space for a desk accessory is fully contained within the startup code, so if a different stack size is required this must be done by modifying the stack size in the startup and reassembling it, the relevant line is:

```
base ds.b 256
```

| | | |
|--------|--------|--|
| -pp | -pp | Generate prototypes with __PROTO for portability |
| -pr | -pr | Generate prototype file |
| -ps | -ps | Generate prototypes only for static functions |
| -qx | -ox | Place quad file in location x |
| -q<n>e | -q<n>e | Quit compilation after n errors/warnings |
| -q<n>w | -q<n>w | Quit compilation after n warnings |
| -q | | Same as -q e w |
| -q- | -q- | Never quit on any errors or warnings |
| -r | -r | Default subroutine calls to near (PC-relative) |
| -r0 | -r0 | Default subroutine calls to far (Absolute) |
| -r1 | -r1 | Default subroutine calls to near (PC-Relative) |
| -rr | -rr | Default subroutine calls/entries to register conventions |
| -rs | -rs | Default subroutine calls/entries to stack conventions |
| -rb | -rb | Generate code for both register and stack convention entries |
| -rx | | Place compiled objects into library x |
| -s | -s | Specify default segment names |
| -sb=x | -sb=x | Specify name for BSS segment |
| -sc=x | -sc=x | Specify name for code segment |
| -sd=x | -sd=x | Specify name for data segment |
| -ta | | Force linking of a desk accessory startup stub |
| -td | | Force linking of auto detecting startup stub |
| -tr | | Force linking of resident startup stub |
| -t=x | | Use file x as the startup code |
| -u | -u | Undefine all preprocessor symbols |
| -ux | -ux | Undefine preprocessor symbol x |
| -v | -v | Disable stack checking code |
| -w | -w | Default to short integers |
| -x | -x | Treat all global declarations as externals |
| -y | -y | Load up A4 with base address at start of functions |
| -z | | Generate GST-linkable code |

which is normally near to the last line in `c.s`.

These stubs perform no command line parsing, no environment setup, no standard file opening and no dynamic memory allocator initialisation. This last point is so that all DAs are forced to conform to the requirement that a DA may not legally use GEMDOS Malloc.

In order to circumvent the malloc problem you may add static arrays to the library free memory list using the `_addheap` function. The function has the prototype:

```
void _addheap(void * base, size_t length);
```

Calling this function with the base of a static array and a length count adds those bytes to the library heap. For example:

```
int main(void)
{
    static long heap[100];
    _addheap(heap, sizeof(heap));
    ...
}
```

Note that the length of the heap *must* be a longword multiple. Also be aware that if you never call this function and your program calls *any* function which must malloc() memory (e.g. `open()`) then all such calls *will* fail.

Because desk accessories are always GEM programs they must always be linked with the GEM library, additionally the DA startup is specified using the `-tA` option.

Auto-detecting - suffix 'aut'

This family of stubs allow a program to detect how it has been run, from the auto-folder, as a desk accessory, as a GEM program or as a TOS program. An external variable, `int_XMODE`, is made available so that the program may perform the requisite initialisation:

| | |
|---|-------------------------|
| 0 | Standard GEM program |
| 1 | Standard TOS program |
| 2 | GEM DA |
| 3 | Auto folder TOS program |

For the GEM/TOS modes (0, 1 and 3) the startup code performs otherwise as described for a normal GEMDOS program. For the DA mode, the stack issue recurs, in this instance the startup code re-uses the space taken by the startup code as the stack for the desk accessory. This gives approximately 750 bytes of stack space, which may be increased by inserting NOPs, immediately before the `isad02` label. Other than this oddity, the DA startup sequence is identical to that described previously.

Resident - suffix 'res'

The resident stubs provide a facility for generating programs which are both residentable and re-entrant. This is used, for example, by LC2 so that it may be multiply re-executed from within the integrated environment. The resident stubs place no special restrictions on the programmer other than that all accesses to the data must be referenced through A4, i.e. the normal -D1 model. Note that these means there must be *no* far data whatsoever.

In the resident mode the startup code initially detects whether or not the program is being executed in a resident manner, or whether it has been run normally, as if by `Pexec(0, ...)`. A copy of the near data section is then made so that any relocation required may be performed on it, prior to initialising A4. Note that in this mode the default locations of any global data variables exported for a symbolic debugger point to the original constant copy of the data and not the copy which is in use.

To use a residentable program from within your own application, it must initially be loaded using `Pexec(3, ...)` and then shrunk viz:

```
#include <osbind.h>
#include <basepage.h>

bp=(BASEPAGE *)Pexec(3,prog,"");
Mshrink(bp, bp->p_tlen+bp->p_dlen+0x100);
Mfree(bp->p_env); /* kill the environment */
```

To execute this program, another basepage must be allocated for the active copy, and then executed:

```
BASEPAGE *ap=(BASEPAGE *)Pexec(5, NULL, command, env);

ap->p_tbase=bp->p_tbase;
ap->p_tlen=bp->p_tlen;
ap->p_dbase=bp->p_dbase;
ap->p_dlen=bp->p_dlen;
ap->p_bbase=ap+1;
ap->p_blen=(long)ap->p_hitpa-(long)ap->p_lowtpa-sizeof(*ap);
err=Pexec(4, NULL, (char *)ap, NULL);
Mfree(ap->p_env);
Mfree(ap);
```

where `command` and `env` give the command line and environment respectively that are to be passed to the child process.

If a residentable program is to be executed normally a standard `Pexec(0, ...)` suffices. Please note that a residentable program whilst conforming to the normal GEMDOS program load format contains many extensions which are generated by the linker in order to support the resident concept. At this time it is *not* our intention to provide detailed information on this since such information is only useful to the startup stubs (provided) and the linker, CLink.

User supplied stubs

A user specified stub may be passed, by the user, to the compiler driver. Typically this is used when the startup code has been modified for a special purpose, or the stack size of a D.A. has been increased, for example.

The name of the stub is communicated to the driver via the `-t=` option, in conjunction with the `-L` options. Hence to compile and link with the stub `mystub.o`:

```
lc -L -t=mystub.o myprog.c
```

Naming conventions

The naming conventions for the startup stubs follows exactly the same pattern as that for libraries. The first character of a startup stub is always `C`, followed by:

| Letter | Type of library |
|-----------------|----------------------------------|
| <code>s</code> | Default short integer (-w) |
| <code>r</code> | Register parameter passing (-fr) |
| <code>nb</code> | Non-base relative (-b0) |

The final three character suffix, as described above, are as follows:

| Suffix | Type of startup stub | Option |
|-------------------|-----------------------------|------------------|
| <code>acc</code> | Desk accessory | <code>-ta</code> |
| <code>aut</code> | Auto program type detecting | <code>-td</code> |
| <code>res</code> | Resident program | <code>-fr</code> |
| <code>User</code> | User specified stub | <code>-t=</code> |

Hence a file called `csrnacc.o` would be a short integer, register passing non-base relative desk accessory startup stub.

Re-assembling c.s

The startup code is provided so that it may be modified as the user requires. To re-assemble it the command:

```
asm [options] c.s
```

should be used, either from a shell, the desktop or as a Tool/Run Other command from the environment.

The options part dictates what sort of stub is to be produced, all types of stub are controlled using various `-O` parameters. These are:

| | |
|----------|---------------------------------|
| AUTO | Auto-detecting (-fd) |
| DA | Desk accessory (-ta) |
| GST | GST compatible (-z) |
| NOBASER | Non-base relative mode (-b0) |
| REGARGS | Register passing mode (-fr) |
| RESIDENT | Residentable program (-fr) |
| SHORTINT | Default short integer mode (-w) |

Hence to build the stub described earlier, `csrnacc.o`, the command used is:

```
asm -dda -dREGARGS -dNOBASER -dSHORTINT c.s
```

Note that this will in fact build the linkable file into `c.o` rather than `csrnacc.o`.

Also note that specifying GST does not build a GST compatible linkable file, the `.o` produced by `asm` must be passed through `lc2gst` first.

Appendix G

ST ASCII Table

Here is the 8-bit ASCII representation of the ST's character set:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|----|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | ! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / | ? |
| 2 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 3 | P | Q | R | S | T | U | V | W | X | Y | Z | [| \ |] | ^ | _ |
| 4 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 5 | p | q | r | s | t | u | v | w | x | y | z | { | | } | ~ | ¯ |
| 6 | ª | á | â | ã | ä | å | Æ | Ö | Ø | Ù | Ú | Û | Ü | Ý | Þ | ß |
| 7 | à | á | â | ã | ä | å | Æ | Ö | Ø | Ù | Ú | Û | Ü | Ý | Þ | ß |
| 8 | ¨ | ª | ¸ | º | » | ¼ | ½ | ¾ | ⅓ | ⅔ | ⅕ | ⅖ | ⅗ | ⅘ | ⅙ | ⅚ |
| 9 | ⅛ | ¼ | ½ | ¾ | ⅓ | ⅔ | ⅕ | ⅖ | ⅗ | ⅘ | ⅙ | ⅚ | ⅛ | ¼ | ½ | ¾ |
| A | ⅓ | ⅔ | ⅕ | ⅖ | ⅗ | ⅘ | ⅙ | ⅚ | ⅛ | ¼ | ½ | ¾ | ⅓ | ⅔ | ⅕ | ⅖ |
| B | ⅗ | ⅘ | ⅙ | ⅚ | ⅛ | ¼ | ½ | ¾ | ⅓ | ⅔ | ⅕ | ⅖ | ⅗ | ⅘ | ⅙ | ⅚ |
| C | ⅙ | ⅚ | ⅛ | ¼ | ½ | ¾ | ⅓ | ⅔ | ⅕ | ⅖ | ⅗ | ⅘ | ⅙ | ⅚ | ⅛ | ¼ |
| D | ⅛ | ¼ | ½ | ¾ | ⅓ | ⅔ | ⅕ | ⅖ | ⅗ | ⅘ | ⅙ | ⅚ | ⅛ | ¼ | ½ | ¾ |
| E | ⅓ | ⅔ | ⅕ | ⅖ | ⅗ | ⅘ | ⅙ | ⅚ | ⅛ | ¼ | ½ | ¾ | ⅓ | ⅔ | ⅕ | ⅖ |
| F | ⅗ | ⅘ | ⅙ | ⅚ | ⅛ | ¼ | ½ | ¾ | ⅓ | ⅔ | ⅕ | ⅖ | ⅗ | ⅘ | ⅙ | ⅚ |

The most significant four bits of the ASCII representation are shown down the left side whereas the least four significant bits are across the top so that, for example:

4C (4*16+12=76 decimal) represents L

7B (7*16+11 = 123 decimal) represents {

Appendix H

VT52 Screen Codes

When writing to the screen via GEMDOS or the BIOS calls, the screen driver emulates VT52 protocols. The control codes are sent via *escape* sequences, which means an escape character is sent (27 decimal, or \$1B) followed by one or more other characters.

| | |
|-------|--|
| ESC A | Cursor up; no effect if at the top line |
| ESC B | Cursor down; no effect if at the bottom line |
| ESC C | Cursor right; no effect if on the right hand side |
| ESC D | Cursor left; no effect if on left hand side |
| ESC E | Clear screen and home cursor |
| ESC H | Home cursor |
| ESC I | Move cursor up one line; if at top scrolls the screen down a line |
| ESC J | Erase to end of screen, from the cursor position onwards |
| ESC K | Clear to end of line |
| ESC L | Insert a line by moving all following lines down. Cursor is positioned at start of the new line |
| ESC M | Delete a line by moving all following lines up |
| ESC Y | Position cursor; should be followed by two characters, the first being the Y position, the second the X. Row and column numbering starts at (32, 32) which is the top left |
| ESC b | Foreground colour; should be followed by a character to determine the colour, of which the four lowest bits are used |
| ESC c | Background colour; similar to above |
| ESC d | Erase from beginning of display to the cursor position |
| ESC e | Enable cursor |
| ESC f | Disable cursor |
| ESC j | Save the current cursor position |
| ESC k | Restore a cursor position saved using ESC j; note that this is not supported on the original 1.0 ROMs |
| ESC l | Erase a line and put cursor at start of line |
| ESC o | Erase from start of line to cursor position |
| ESC p | Inverse video on |
| ESC q | Inverse video off |
| ESC v | Wrap around at end of line on |
| ESC w | Wrap around at end of line off |

For instance these codes may be used in a normal printf statement to print information one after another, on the same line, e.g.

```
printf("\033IProcessing %s\033K\n",file);
```

Appendix I Bibliography

C Programming

Advanced C **Schildt, Herbert (1986)**
ISBN 0-07-881208-9, McGraw-Hill, Berkeley, CA 94710, USA.

Advanced C Techniques & Applications
Sobelman, Gerald E. and David E. Krekelberg (1985)
ISBN 0-88022-162-3, QUE Corporation, Indianapolis, IN 46250, USA.

Advanced C: Food For The Educated Palate **Gehani, Nardin (1985)**
ISBN 0-88175-078-6, Computer Science Press, Rockville, MD 20850, USA.

ANSI C: A Lexical Guide **Mark Williams Company (1988)**
ISBN 0-13-037814-3, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

ANSI X3.159-1989 - Programming Language - C **ANSI (1990)**
American National Standards Institute, 1430 Broadway, New York 10018, USA.

C For Beginners **Sinclair, Ian (1986)**
ISBN 0-86161-206-X, Melbourne House Publishers Ltd., 60 High Street, Kingston-Upon-Thames, Surrey KT1 4DB, U.K.

C Programming Guide **Purdum, Jack (1983)**
ISBN 0-88022-022-8, QUE Corporation, Indianapolis, IN 46250, USA.

C Self-Study Guide **Purdum, Jack (1985)**
ISBN 0-88022-149-6, QUE Corporation, Indianapolis, IN 46250, USA.

C Wizard's Programming Reference **Schwaderer, W. David (1985)**
ISBN 0-471-82641-3, Wiley Press, New York, NY 10158, USA.

C: A Reference Manual, 2nd Edition
Harbison, Samuel P. and Guy L. Steele Jr (1984)
ISBN 0-13-109802-0, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

Common C Functions

ISBN 0-88022-069-4, QUE Corporation, Indianapolis, IN 46250, USA.

Debugging C

ISBN 0-88022-261-1, QUE Corporation, Indianapolis, IN 46250, USA.

Efficient C

ISBN 0-911537-05-8, Plum Hall, Cardiff, NJ 08232, USA.

Going from BASIC to C

ISBN 0-13-357799-6, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

Introducing C

ISBN 0-00383-105-1, William Collins Sons & Co. Ltd., 8 Grafton Street, London W1X 3LA, U.K.

Learning to Program in C

ISBN 0-911537-00-7, Plum Hall, Cardiff, NJ 08232, USA.

Microsoft C Run-Time Library

ISBN 1-55615-227-2, Microsoft Press, Bellevue, WA 98009, USA.

Programming in C For The Microcomputer User

ISBN 0-13-729641-X, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

Software Engineering in C

ISBN 0-387-96574-2, Springer-Verlag, 175 Fifth Avenue, New York, NY 10010, USA.

Standard C

ISBN 1-55615-158-6, Microsoft Press, 16011 NE 36th Way, Box 97017, Redmond, Washington, 98073-9717, USA.

The C Compendium

ISBN 0-946498-86-6, Sunshine Books, 12-13 Little Newport Street, London WC2H 7PP, U.K.

The C Library

ISBN 0-07-881110-4, McGraw-Hill, Berkeley, CA 94710, USA.

The C Programmer's Handbook

ISBN 0-13-110073-4, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

The C Programmer's Handbook

ISBN 0-89303-365-0, Brady Communications Company, Inc., Bowie, MD 20715, USA.

The C Programming Language, 2nd Edition

ISBN 0-13-110370-9, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

The C Toolbox

ISBN 0-201-11111-X, Addison-Wesley Publishing Company, Reading, MA, USA.

UNIX System V Programmers Reference Manual

ISBN 0-13-940479-1, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

Variations In C

ISBN 0-914845-48-9, Microsoft Press, Bellevue, WA 98009, USA.

68000

16-Bit Microprocessors

ISBN 0-00-383113-2, William Collins Sons & Co. Ltd., 8 Grafton Street, London W1X 3LA, U.K.

68000 Assembly Language Programming 2nd Edition

ISBN 0-07-881232-1, Osborne/McGraw-Hill, 2600 Tenth Street, Berkeley, CA 94710, USA.

68000 Machine Code Programming

ISBN 0-00-383163-9, William Collins Sons & Co. Ltd., 8 Grafton Street, London W1X 3LA, U.K.

68000, 68010, 68020 Primer

ISBN 067-22405-4, Howard W.Sams & Co., 4300 W.62nd Street, Indianapolis, IN 46268, USA.

Brand, Kim Jon (1985)

ISBN 0-88022-069-4, QUE Corporation, Indianapolis, IN 46250, USA.

Ward, Robert (1986)

ISBN 0-88022-261-1, QUE Corporation, Indianapolis, IN 46250, USA.

Plum, Thomas and Jim Brodie (1985)

ISBN 0-911537-05-8, Plum Hall, Cardiff, NJ 08232, USA.

Traister, Robert J. (1985)

ISBN 0-13-357799-6, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

Allen, Boris (1986)

ISBN 0-00383-105-1, William Collins Sons & Co. Ltd., 8 Grafton Street, London W1X 3LA, U.K.

Plum, Thomas (1983)

ISBN 0-911537-00-7, Plum Hall, Cardiff, NJ 08232, USA.

Jamsa, Kris (1985)

ISBN 1-55615-227-2, Microsoft Press, Bellevue, WA 98009, USA.

Traister, Robert J. (1984)

ISBN 0-13-729641-X, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

Darnell, Peter A. and Philip E. Margolis (1988)

ISBN 0-387-96574-2, Springer-Verlag, 175 Fifth Avenue, New York, NY 10010, USA.

Plauger, P. L. and J. Brodie (1989)

ISBN 1-55615-158-6, Microsoft Press, 16011 NE 36th Way, Box 97017, Redmond, Washington, 98073-9717, USA.

Lawrence, David and Mark England (1985)

ISBN 0-946498-86-6, Sunshine Books, 12-13 Little Newport Street, London WC2H 7PP, U.K.

Jamsa, Kris (1985)

ISBN 0-07-881110-4, McGraw-Hill, Berkeley, CA 94710, USA.

M68000 Family Programmer's Reference Manual

Motorola Semiconductor Products Inc., PO Box 20912 Phoenix, AZ 85036, USA.
Motorola Inc. (1989)

Mastering The 68000 Microprocessor Robinson, Phillip R. (1985)
ISBN 0-8306-1886-4, Tab Books Inc., Blue Ridge Summit, PA 17214, USA.

Microprocessor Systems: A 16-Bit Approach Eccles, William J. (1985)
ISBN 0-201-11985-4, Addison-Wesley Publishing Company, Reading, MA, USA.

Programming the 68000 Williams, Steve (1985)
ISBN 0-89588-133-0, SYBEX Inc., 2021 Challenger Drive #100, Alameda, CA 94501, USA.

The MC68000 User's Manual 7th Edition Motorola Inc. (1989)
ISBN 0-13-567074-8, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

The MC68020 User's Manual 2nd Edition Motorola Inc. (1985)
ISBN 0-13-566878-6, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

The MC68030 User's Manual Motorola Inc. (1987)
Motorola Semiconductor Products Inc., PO Box 20912 Phoenix, AZ 85036, USA.

The MC68881/MC68882 User's Manual Motorola Inc. (1987)
ISBN 0-13-566936-7, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, USA.

Algorithms & Data Structures

Algorithms in C

ISBN 0-201-51425-7, Addison-Wesley Publishing Company, Reading, MA, USA.
Sedgewick, Robert (1990)

C Chest and Other C Treasures

ISBN 0-934375-40-2, M & T Books, 501 Galveston Drive, Redwood City, CA 94063, USA.
Holub, Allen I. (1987)

Compilers: Principles, Techniques and Tools

ISBN 0-201-10194-7, Addison-Wesley Publishing Company, Reading, MA, USA.
Aho, Alfred V, Ravi Sethi and Jeffrey D. Ullman (1986)

Data Handling Utilities in C Radcliffe, Robert A. and Thomas J. Raab (1986)
ISBN 0-89588-304-X, Sybex, Berkeley, CA 94710, USA.

Data Structures and Algorithms Aho, Alfred V, John E. Hopcroft and Jeffrey D. Ullman (1983)
ISBN 0-201-00023-7, Addison-Wesley Publishing Company, Reading, MA, USA.

Fundamental Algorithms Knuth, Donald E. (1973)
ISBN 0-201-03809-9, Addison-Wesley Publishing Company, Reading, MA, USA.

Seminumerical Algorithms Knuth, Donald E. (1981)
ISBN 0-201-03822-9, Addison-Wesley Publishing Company, Reading, MA, USA.

Sorting and Searching Knuth, Donald E. (1973)
ISBN 0-201-03803-X, Addison-Wesley Publishing Company, Reading, MA, USA.

ST Specific

A Hitchhikers Guide to the BIOS Atari Corp (1986)
Atari Corp, 1196 Borregas Avenue, Sunnyvale, CA 94086, USA.

Atari GEMDOS Reference Manual Atari Corp (1986)
Atari Corp, 1196 Borregas Avenue, Sunnyvale, CA 94086, USA.

Atari ST Internals 3rd Edition Brückmann, Rolf, Lothar Englisch and Klaus Gerits (1988)
ISBN 0-916439-46-1, Data Becker GmbH, Merowingerstraße 30, 4000 Düsseldorf, West Germany.

COMPUTE!'s ST Applications Guide: Programming in C Field, Simon, Kathleen Mandis and Dave Myers (1986)
ISBN 0-87455-078-5, COMPUTE! Publications, Inc., P.O. Box 5406 Greensboro, NC 27403, USA.

COMPUTE!'s Technical Reference Guide, Atari ST - Volume I: VDI
Sheldon Leeman (1987)
NC 27403, USA.

Concise Atari ST 68000 Programmer's Reference

ISBN 1-85181-017-X, Glentop Publishers Ltd., Standfast House, Bath Place,
High Street Barnet, Herts EN5 5XE, U.K.

GEM Programmer's Guide, Volume 1: VDI Digital Research (1987)
Digital Research Inc., 60 Garden Court, P.O. Box DRI, Monterey, CA 93942,
USA.

GEM Programmer's Guide, Volume 2: AES Digital Research (1987)
Digital Research Inc., 60 Garden Court, P.O. Box DRI, Monterey, CA 93942,
USA.

GEMDOS Extended Argument (ARGV) Specification Atari Corp (1989)
Atari Corp, 1196 Borregas Avenue, Sunnyvale, CA 94086, USA.

Professional GEM
ANTIC Publishing

Oren, Tim (1985)

Programmers Guide to GEM Balma, Phillip and William Fittler (1986)
ISBN 0-89588-297-3, SYBEX Inc., 2344 Sixth Street, Berkeley, CA 94710, USA.

Rainbow TOS Release Notes

Atari Corp, 1196 Borregas Avenue, Sunnyvale, CA 94086, USA.

S.A.L.A.D. - Still Another Line A Document Atari Corp (1987)
Atari Corp, 1196 Borregas Avenue, Sunnyvale, CA 94086, USA.

STE TOS Release Notes

Atari Corp, 1196 Borregas Avenue, Sunnyvale, CA 94086, USA.

The Pexec Cookbook

Atari Corp, 1196 Borregas Avenue, Sunnyvale, CA 94086, USA.

Appendix J Technical Support

So that we can maintain the quality of our technical support service we are detailing how to take best advantage of it. These guidelines will make it easier for us to help you, fix bugs as they get reported and save other users from having the same problem. Technical support is available in five ways:

Phone our technical support hour is normally between 3pm and 4pm, though non-European customers' calls will be accepted at other times.

Post if sending a disk, please put your name & address on it.

BIX™ our username is (not surprisingly) *hisoft*. Would UK customers please use CIX or more old fashioned methods; it's cheaper for everyone.

CIX™ our username is (still not surprisingly) *hisoft*.

GEnie™ our username is ... OK, you've guessed it!

For bug reports, please always quote the program, computer, version number of the program (the one from the file READ.ME) and the serial number found on your master disk. If your problem is with the compiler please ensure that you tell us *all* compiler options which you used so that we can attempt to duplicate the problem.

If you think you have found a bug, try and create a small program that reproduces the problem. It is always easier for us to answer your questions if you send us a letter and, if the problem is with a particular source file, please enclose a copy on disk (which we *will* return).

Remember, technical support is *only* available to registered users i.e. those people that have filled out the registration card enclosed with Lattice C 5 and mailed it to HiSoft.

Upgrades

As with all our products, Lattice C 5 is undergoing continual development and, periodically, new versions become available. We normally make a small charge for upgrades, though if extensive additional documentation is supplied the charge may be higher. All users who return their registration cards will be notified of *major* upgrades.

Suggestions

We welcome any comments or suggestions about our programs and, to ensure we remember them, they should be made in writing.

Common Problems

The following is a list of common mistakes made when starting programming with Lattice C. Please check these carefully against your problem before ringing in for technical support.

- ? Why does the linker complain that symbols beginning with `__CX` or `__AES` are undefined and wants me to enter a "DEFINE value"?
- These are symbols in the floating point maths library and/or GEM libraries (respectively), if you have used any floating point maths or GEM calls you must link with the relevant library. This involves selecting Link with floating point or Link with GEM from the options menu if using the integrated compiler, or using the `-Lm` or `-Lg` options from the command line.
- ? My program was nearly finished, but suddenly the compiler has started reporting "Intermediate file error". How can I stop this?
- This almost certainly means that the compiler has run out of space on the device on which it is creating the quad (intermediate) file. Either delete some files from the appropriate disk, or if it is a RAM disk, and you have enough memory left, increase its size.
- ? Whenever I try to read/write screen images, they always get corrupted. Why?
- The libraries support two modes of file operation, text (the default) and binary. In text mode all CR/LF pairs are considered special and may be removed/added by the library. If using a binary file (such as a screen image) make sure you have selected the binary mode of operation, see the page on `fopen()` etc. for more details on this.
- ? My program mysteriously crashes, and I think this is the compiler corrupting the stack. Is this a bug?
- What has probably happened is that you have either allocated too much stack space with stack checks disabled (these are *on* by default), or that you are writing over the end of an array which you have allocated on the stack. This problem can be hard to track down, so always make sure you test your program with stack checks on.

Index

- \$ legality 61
- 68000 68, 217
- 68010 68
- 68020 68, 217
- 68030 68, 217
- 68881 floating point 63
- enum 75
- signed 76
- void 76
- volatile 77
- APPLBLK 290, 291
- ASCII Table 337
- ASCII Table, Editor 42
- ASCII, assembler Literal 212
- ASM, the assembler 3, 211
- Assembler 3, 211
- calling conventions 228
- Comment 214
- conditional assembly 223
- conditionals
 - end, ENDC 223
 - if defined, IFD 223
 - if equal, IFEQ 223
 - if equivalent, IFC 223
 - if greater than or equal, IFGE 223
 - if greater than, IFGT 223
 - if less than or equal, IFLE 223
 - if less than, IFLT 223
 - if not defined, IFND 223
 - if not equivalent, IFNE 223
 - toggle, ELSE 223
- constants 212
- data listing 217
- directives 218
 - CNOP, conditional alignment 218
 - conditional assembly 223
 - CSECT 226
 - CSECT, conditional alignment 218
 - DC, define constants 219
 - DS, define space 220
 - ELSE, toggle conditional assembly 223
 - END, end assembly 220
 - ENDC, end conditional assembly 223
 - ENDM, end macro definition 220
 - EQU, equate label 220
 - IDNT 220
 - IFC, if equivalent 223
 - IFD, if defined 223
 - IFEQ, if equal 223
 - IFGE, if greater than or equal 223
 - IFGT, if greater than 223
 - IFLE, if less than or equal 223
- abandon file, resource editor 162
- absolute code 71, 119
- absolute expression, assembler 213
- Adding symbols 66
- Additional Tools 249
- address error 206
- Addressing modes 214
- aggregate equivalence, strengthen, -cq 61
- Alert boxes, debugger 178
- Alert, resource editor 169
- align long objects 67
- legality 61
- 68000 68, 217
- 68010 68
- 68020 68, 217
- 68030 68, 217
- 68881 floating point 63
- asm keyword 79, 233
- BSSBAS, linker 120
- BSSLEN, linker 120
- DATABAS, linker 120
- DATALEN, linker 120
- DATE, pre-processor symbol 58
- edata, linker 121
- end, linker 121
- etext, linker 121
- far keyword 77
- FILE, pre-processor symbol 58
- huge keyword 77
- interrupt keyword 79
- LINE, pre-processor symbol 58
- LinkerDB, linker 120
- MERGED, linker 115
- near keyword 78
- PLAIN_CHAR_UNSIGNED pre-processor symbol 59
- regargs keyword 71, 79, 233
- REBASE, RESLEN, linker 120
- saveds keyword 73, 79
- sidargs keyword 71, 80
- STDC pre-processor symbol 58
- TIME, pre-processor symbol 58
- OSERR 32
- abandon file, resource editor 162
- absolute code 71, 119
- absolute expression, assembler 213
- Adding symbols 66
- Additional Tools 249
- address error 206
- Addressing modes 214
- aggregate equivalence, strengthen, -cq 61
- Alert boxes, debugger 178
- Alert, resource editor 169
- align long objects 67

A

IFLT, if less than 223
 IFNC, if not equivalent 223
 IFND, if not defined 223
 INCBIN, include binary 220
 INCLUDE, include source 220
 LIST, enable listing 221
 MACRO, define macro 221
 MEXIT, exit macro invocation 221
 NOLIST, disable listing 221
 OFFSET, start offset section 221
 PAGE, page throw 222
 RORG, relocatable origin 222
 SECTION, define program section 222
 SET, temporary equate 222
 TTL, set title 222
 XDEF, export label 222
 XREF, import label 222
 error messages 239
 Function Exit Rules 234
 includes, listing 217
 Label 211
 labels
 equate, EQU, = 220
 start offset section, OFFSET 221
 temporary equate, SET 222
 listing
 disable 221
 enable 221
 page throw 222
 title 222

macro expansion listing 217
 macros
 calling 225
 default parameters 224
 define macro, MACRO 221
 definition 224
 end macro definition, ENDM 220
 exit macro invocation, MEXIT 221
 Number Representations 212
 Opcode 212
 Operands 212
 Operation 212
 operator 213
 options 216
 debugging, -d 216
 defining symbols, -d 216
 listing 217
 data generation 217
 include files 217
 macro expansion 217
 object file, placing, -o 216
 short integer, -w 217
 target processor, -m 217
 underline prefacing, -u 217
 reserved symbols
 NARG, number of macro arguments 221

variable 212
 assembler, ASCII literal 212
 assembler, binary number 212
 assembler, decimal number 212
 assembler, floating point 215
 assembler, hexadecimal number 212
 assembler, octal number 212
 assembler, running 216
 Assembly language 225
 calling conventions 79, 80, 233
 resource editor 294
 assembly language calling
 conventions 79
 ATARI pre-processor symbol 58
 auto naming, resource editor 149, 162
 auto size, resource editor 162
 Auto Snap, resource editor 152, 162
 auto-detecting program 332
 Automatic Link Vector 119
 Automatic linking 66, 72
 automatic program detection 72
 automatic registration 68
 AVAIL, CLI 129

B

Backspace key, Editor 28
 Backup, master disks 1
 Backups, editor 31
 base-relative 59, 78, 119, 228
 BASIC, resource editor 294
 Batch file 139
 comments 136
 exit 134
 parameters 139
 batch mode 66
 Batchter 2, 129, (See CLI)
 Beginning of line, Editor 27
 Bibliography 341
 68000 343
 Algorithms & Data Structures 344
 C Programming 341
 ST Specific 345
 Big compiler 59
 Binary, assembler 212
 BITBLK structure 289
 Block buffer, Editor 35
 Block commands (see Editor, block
 commands)

Border, resource editor (see Resource
 Editor, border)
 Box, resource editor 144
 BoxChar, resource editor 144, 150
 BoxText, resource editor 144
 breakpoint, debugger (see Debugger,
 breakpoint), (see Debugger,
 breakpoint)
 building pre-compiled headers 74
 Built-in Functions 80
 bus error 206
 Button, resource editor 144

C

C++ mode 60
 calling assembler macros 225
 calling assembly language 228
 Calling Conventions 78
 assembly language 233
 assembly language 79
 interrupt code 79
 register 79, 233
 standard 80
 cancel, resource editor 161
 Case dependency, Editor 29
 CD, CLI 129
 char, unsigned 59, 62
 character constants, multiple, -cm 61
 character set, extended 63
 Child number, resource editor 159
 CHK instruction 206
 choosing a library 121
 Clear text, Editor 30
 CLI 129
 AVAIL 129
 Batch files 139
 parameters 139
 CD 129
 clear screen 130
 CLS 130
 COLOUR 130
 comments 136
 COPY files 131
 COPYWARN 132
 create directory 135
 Cursor keys 138
 DC 132
 DEL 132
 delete

directory 136
 files 132
 DIR 133
 directory
 change 129
 create 135
 current 129
 delete 136
 list 133
 disk change 132
 disk format 134
 disk free space 135
 DISKCHANGE mode 133
 drive, current 130
 ECHO commands 134
 enable file overwrite warnings 132
 environment variables, setting 137
 ERA 134
 EXIT batch file 134
 font 137
 FORMAT disk 134
 FREE disk space 135
 history 138
 Line Editing 138
 list
 directory 133
 file 137
 memory free 129
 MKDIR 135
 MOUSE control 135
 PAUSE command 136
 redirection 140
 REM 136
 REN 136
 rename file 136
 RMDIR 136
 SCREENSAVE mode 136
 SET 137
 SMALL characters 137
 TYPE 137
 VIRTUOLDISK 137
 WHICH 138
 CLink, The Linker 3, 23, 113
 CLINKWITH environment variable
 119
 CLS, CLI 130
 CNOP directive, assembler 218
 co-processor, maths 63
 code generation 68
 68000, -m0 68
 68010, -m1 68

68020, -m2 68
 68030, -m3 68
 automatic registration, -mr 68
 deferred stack cleanup, -mc 68
 family mode, -ma 68
 short integer, -w 73, 217
 space optimisation, -ms 68
 stack checking, -v 73
 time optimisation, -mt 69
 code generator 53
 code model 71, 228
 COLOUR, CLI 130
 colour, resource editor (see Resource Editor, colour)
 Command Line Interpreter (See CLI) Editor 38
 COMMAND.COM 129
 comment, assembler 214
 compatibility mode
 \$ legality, -cd 61
 allow register keywords, -cr 61
 ANSI, -ca 60
 C++, -c+ 60
 constant string copying, -cs 61
 enable structure warnings, -ct 62
 enforce prototyping, -cf 61
 error source line suppression, -ce 61
 longword align data, -cl 61
 make char unsigned, -cu 62
 make globals external, -cx 62
 multiple character constants, -cm 61
 permit new keywords, -ck 61
 pre-ANSI pre-processor, -co 61
 return value warnings, -cw 62
 strengthen aggregate equivalence, -cq 61
 suppress includes, -ci 61
 compatibility option 60
 Compile, Editor 36
 Compiler 2, 51
 built-in functions 80
 Calling Conventions 78
 character set 63
 code generator 53
 Command line operation 51
 compiling and linking from Editor 37
 compiling from Editor 36
 CXERRs 110
 Environment Variables 56

Errors 84
 Extensions 74
 ANSI 75
 const 75
 enum 75
 signed 76
 void 76
 volatile 77
 Calling Conventions 78
 _asm 79, 233
 _interrupt 79
 _savesd 79
 Storage Classes 77
 far 77
 huge 77
 near 78
 file name 58
 goto error 36
 Integrated 25
 Internal Errors 110
 lc1 52
 lc1b 52
 LC2 36, 53
 librarian 72
 line number 58
 linking 66, 72
 options 39, 59
 options, Tutorial 18
 Parser 52
 phase 1 52
 phase 2 53
 phases 52
 Pre-compiled Header Files 74
 Pre-processor 52
 Return Codes 51
 start date 58
 start time 58
 Storage Classes 77
 syntax check 36
 compiler includes
 cross reference 64
 listing 64
 Compiler options, saving in Editor 48
 Compiler Tools 249
 Compiling and linking, Tutorial 6, 9
 compress header files 251
 conditional alignment, assembler, CNOP 218
 conditional alignment, assembler, CSECT 218
 conditional assembly see assembler

const modifier 75
 constant strings 61
 constants, assembler 212
 continuous compilation 60
 control characters, resource editor 150
 controlling errors 65
 controlling warnings 65
 converting programs 305
 Copy block, Editor 34
 COPY files, CLI 131
 copy memory, debugger 201
 copy, resource editor 161
 COPYWARN, CLI 132
 cross referencing 64, 217
 compiler includes, -gc 64
 linker symbols, -Lx 67
 symbols, -gx 65
 cross referencing, linker 117
 CSECT directive 226
 CSECT directive, assembler 218
 Cursor appearance, Editor 44
 Cursor keys
 CLI 138
 debugger 189
 Editor 27
 Cursor positioning, Editor 26
 cut, resource editor 161
 CXERR, compiler 110

D

data model 59, 77, 119, 228
 data, longword align 61
 DC directive, assembler 219
 DC, CLI 133
 debug
 suppression with linker 116
 DEBUG pre-processor symbol 59
 DEBUG symbol 62
 Debugger 3, 22
 abort 194
 address, set 187
 alert boxes 178
 Auto Load Source 200
 Automatic Prefix Labels 200
 baseconvert 188
 breakpoint 181, 191
 conditional 191
 count 191

GEMDOS 193
 kill 193
 permanent 191
 remove 193
 set 187, 192, 193
 show 192
 simple 191
 stop 191
 command summary 202
 compiling 176
 copy memory 201
 cursor keys 189
 dialog boxes 178
 disassembly window 186
 edit 187
 executing programs 195
 expressions 182
 find 197
 Follow Traps 199
 font 187
 front panel 179
 GEMDOS breakpoint 193
 hints 204
 history 193
 Ignore Case 199
 input 180
 interrupt program 190
 labels 184
 line numbers 181, 199
 linker symbols, Editor 40
 load binary 194
 load program 194
 load source 195
 lock window 188
 low resolution 186, 190, 195
 memory layout 207
 memory window 186
 MonSTZC 175
 numbers 183
 overview 181
 preferences 198
 print window 188
 register
 set 188
 window 185
 registers 184
 Relative Offsets 199
 running 177
 running program 196
 go 196
 instruction 196
 slowly 196

until 196
 running programs 195
 save binary 195
 Save preferences 201
 screen mode 190
 screen switching 189, 198
 search memory 197
 Show Line Numbers in Source 199
 single step 195
 skip 196
 split window 188
 symbols 184
 Symbols Option 200
 terminate 194
 Top Of RAM 200
 user screen 189
 windows 180, 185
 commands 187
 disassembly 186
 edit 187
 lock 188
 memory 186
 print 188
 register 185
 source code 187
 split 188
 type 189
 zoom 189
 zoom window 189
 debugging mode 62
 debugging, assembler option 216
 debugging, removing 66, 258
 Decimal, assembler 212
 default assembler macro parameters 224
 Default, flag type 286
 deferred stack cleanup 68
 define assembler macro, MACRO 221
 define constants, assembler, DS 220
 define pre-processor symbols 63
 define program section, assembler, SECTION 222
 define space, assembler, DS 220
 defines, cross reference 64
 defining assembler macros 224
 defining symbols, assembler 216
 DEGAS 260
 DEL, CLI 132
 Delete all text, Editor 30
 Delete block, Editor 35
 Delete key, Editor 28

Delete line, Editor 30
 Delete to end of line, Editor 30
 delete, resource editor 162
 Deleting text, Editor 30, 32
 denormalised 266, 267
 Desk accessories, Editor 49
 Desk accessory
 building 331
 linking 72
 Saved! 50
 Desktops, resource editor 292
 Dialog boxes, debugger 178
 Digital Research RCS 259
 DIR, CLI 133
 Directory change, Editor 32
 disassemble, object module, omd 252
 disable debugging 62
 disable stack checking 73
 disassembler, object module, omd 252
 Disk Operations, Editor 31
 DISKCHANGE, CLI 134
 double precision 267
 double precision, force 64
 DRI linkable code 313
 DS directive, assembler 220

E
 ECHO, CLI 134
 EdC, The Screen Editor 2, 25
 EDC.PRG 25
 LC.PRG 25
 EDC.INF 50
 EDC.PRG 4
 EDCTOOLS.INF 9, 45, 48
 Editable, flag type 286
 Editing Icons, resource editor 154
 Editing Images, resource editor 152
 Editor 2, 25
 arrange windows 40
 ASCII table 42
 automatic indent 43
 backspace key 28
 backups 31, 33, 43
 beginning of line 27
 block commands 34
 block buffer 35
 block end 34

block markers 34, 35
 block start 34
 copy block 34
 copy to block buffer 35
 cut & paste between windows 48
 delete block 35
 marking a block 34
 paste block 35
 print block 35
 remember block 35
 save block 34
 change directory 32
 changing fonts 41
 compiler options, saving 48
 compiling and linking C 36
 compile 36
 compile & link 37
 Compiler options 39
 global optimiser 40
 goto error 36
 jump to error 36
 link with floating point 40
 link with GEM 40
 linker symbols 40
 linking 37
 load LC2 43
 running the program 37
 running the program under GEM 37
 syntax check 36
 thorough check 36
 cursor appearance 43
 cursor keys 27
 cursor positioning 26
 cut & paste blocks 48
 cycle windows 41
 delete all text 30
 delete file 32
 delete key 28
 delete line 30
 delete to end of line 30
 deleting text 30
 desk accessories 49
 disk operations 31
 EDC.PRG 25
 end of line 27
 Find (See Search)
 fonts, changing 41
 goto end of file 29
 goto line 28
 goto top of file 28
 Help key 43, 48
 inserting text 32
 LC.PRG 25
 load another command 48

load automatically 49
 loading text 31
 making backups 31
 numeric keypad 47
 numeric pad 43
 options menu 39
 page down 27
 page up 27
 PATH and Saved! 50
 preferences command 36, 37, 43
 preferences, saving 45
 quit 33
 replace all 30
 replacing 29
 run other 37, 38
 run with shell 38
 save as... 31
 save preferences 45
 save text 31
 search
 case dependency 29
 control characters 30
 next 29
 previous 29
 special characters 30
 Tab 30
 searching 29
 searching and replacing 29
 smart parentheses 43
 start of line 27
 status line 26
 tab key 27
 tab size, changing 43
 text buffer, changing 43
 Tools 45
 command line 47
 configuring 46
 environment 47
 errors on return 47
 installing new tool 46
 path 46
 pause after running tool 47
 run other/run with shell 46
 running tool 47
 save files before running tool 47
 saving info on tools 48
 type of tool 46
 undelete line 30
 windows, arrange 40
 windows, cycle 41
 windows, description 49
 windows, switching between them 48

word left 27
 word right 27
 WordStar keys 27
 wrap at end of line 43
 ELSE directive, assembler 223
 empty type 76
 enable debugging 62
 enable listing, LIST 221
 enable structure warnings 62
 Enabling warnings 66
 end assembler macro definition, ENDM 220
 END directive, assembler 220
 End of file, goto, Editor 29
 End of line, Editor 27
 ENDC directive, assembler 223
 ENDM directive, assembler 220
 enforce prototyping 61
 entry rules 231
 enum keyword 75
 enum type 75
 Environment
 CLI 137
 CLINKWITH variable 119
 Compiler 56
 INCLUDE variable 36
 LIB variable 114
 PATH variable 38, 39
 saving variables used by tools, Editor 48
 setting 56, 63
 variables
 INCLUDE, include path 56
 LC.OPT, Default options 57
 LIB, Library path 57
 PATH, Executable path 56
 QUAD, Intermediate directory 57
 variables used by tools, Editor 47
 environment variables 137
 EQU directive, assembler 220
 equate label, assembler, EQU, = 220
 ERA, CLI 134
 Errors 22, 84
 control 65
 maximum 70
 source line suppression 61
 Tutorial 11
 Exceptions, re-install, debugger 202
 excluded source listing 64
 Executable path (See Environment, PATH)

executing programs, debugger 195
 exit macro invocation, assembler, MEXIT 221
 Exit rules 234
 EXIT, CLI 134
 Exit, Editor 33
 Exit, flag type 286
 expert level, resource editor 165
 exponent 266, 267
 export assembler symbol, XDEF 222
 expressions, debugger 182
 extended character set 63
 extended debug, option linker 116
 Extended Type, resource editor 159
 Extensions, language 74
 external, make globals 62, 73
 extract prototypes 70
 Extras, resource editor 152, 159

F
 far keyword 59, 71, 77
 FBoxText, resource editor 144, 150
 file name field width, linker 118
 file selector 5, 8
 fill, resource editor (see Resource Editor, fill)
 find name, resource editor 163
 Find next, Editor 29
 Find previous, Editor 29
 find text, resource editor 163
 find, debugger 197
 Find, Editor (See Editor, searching)
 Flag States (See Resource Editor, states)
 flags, resource editor 158, 286
 Floating point 63
 auto-detecting 63
 double-precision format 267
 I/O mapped 63
 IEEE 64
 linking with, Editor 40
 MC68881 63
 Single-precision format 266
 floating point, assembler, 215
 Font
 changing in CLI 137
 changing in Editor 41
 debugger 187

converter 325
 GenST 313
 librarian 322
 limitations 314
 linker 313
 command line 314
 control files 317
 error messages 319
 warnings 319
 GSTlib (see GST, librarian)

H
 Half Character Snap, resource editor 152, 163
 header file compressor, lcompact 251
 Header files, locating 56, 65
 Header files, pre-compiled (See Pre-compiled header)
 Height, resource editor 159
 Hexadecimal, assembler 212
 Hide, flag type 286
 Hints
 debugger 204
 general 21
 resources 291
 HiSoft C 311
 history
 CLI 138
 debugger 193
 hramdisk 249
 HRD file
 format 301
 resource editor 146
 huge keyword 77
 hunk field width, linker 118
 Hunk map 66

I
 IBox, resource editor 144
 Icon
 resource editor 144
 editing 154
 importing 156, 260
 ICONBLK structure 289
 identifier significance 69
 IDNT directive, assembler 220
 IFC directive, assembler 223
 IFD directive, assembler 223

FORMAT, CLI 135
 forms, resource editor 166
 FORTRAN, resource editor 295
 Free Image, resource editor 171
 Free String, resource editor 168
 FRBE, CLI 135
 from files, linker 114
 FText, resource editor 144, 150
 Function
 entry rules 231
 exit rules 234
 return value 62

G
 G_BOX 284
 G_BOXCHAR 285
 G_BOXTEXT 285
 G_BUTTON 285
 G_FBOXTEXT 286
 G_FTEXT 285
 G_IBOX 285
 G_ICON 286
 G_IMAGE 285
 G_PROGDEF 285
 G_STRING 285
 G_TEXT 285
 G_TITLE 286
 GEM
 changing fonts, Editor 41
 example, Tutorial 16
 linking 66
 running a compiled program 37, 40
 tool type, editor 46
 generic pointer 76
 GenST 313
 Global optimiser
 Editor 40
 invoking 69
 Global optimiser, GO 3, 53
 globals data register 73, 79
 globals, make external 62, 73
 Goto end of file, Editor 29
 Goto error, Editor 36
 Goto error, Tutorial 14
 Goto line, Editor 28
 Goto top of file, Editor 28
 GST
 compiler option 73
 compiling 314

IFEQ directive, assembler 223
 IFGE directive, assembler 223
 IFGT directive, assembler 223
 IFLE directive, assembler 223
 IFLT directive, assembler 223
 IFNC directive, assembler 223
 IFND directive, assembler 223
 illegal instruction 206
 Image
 resource editor 144
 editing 152
 importing 156, 260
 Implementation Behaviour 263
 import assembler symbol, XREF 222
 Importing Images 260
 resource editor 156
 INCBIN directive, assembler 220
 INCLUDE (See Environment,
 INCLUDE)
 include binary, assembler, INCBIN
 220
 INCLUDE directive, assembler 220
 include files
 locating, compiler 65
 resource editor 145
 suppression 61
 Include path (See Environment,
 INCLUDE)
 include source, assembler, INCLUDE
 220
 INCLUDE, set path 65
 Index in tree, resource editor 159
 INDIRECT, flag type 287
 Infinity 266, 267
 Inline
 directive 82
 OS Calls 82
 Inserting text, Editor 32
 Install application 49
 Installation guide 4, 9
 Integrated Compiler 25
 Intermediate directory (See
 Environment, QUAD)
 Intermediate files 22
 placing 70
 Intermediate files, locating 57
 interrupt code conventions 79
 interrupt program, debugger 190
 Invalid block! 34

J

Jump to error, Tutorial 14

K

K-RSC 259
 Keywords 74
 asm 79, 233
 far 77
 huge 77
 interrupt 79
 near 78
 regargs 71, 79
 saveds 73, 79
 stdargs 71, 80
 const 75
 enum 75
 far 59, 71, 77
 huge 77
 near 59, 71, 78
 permit new keywords 61
 signed 76
 void 76
 volatile 77
 keywords, linker 116

L

labels, debugger 184
 Language Extensions 74
 Language, resource editor 157
 large code model 71, 119
 large data model 59, 77, 119
 LASTOB, flag type 286
 Lattice 3.04 305
 LATTICE pre-processor symbol 58
 LATTICE_50 pre-processor symbol 58
 LC, The compiler 2, 51
 LC.PRG
 Integrated Compiler 25
 LC1 3, 52
 lc1b 52
 LC2 3, 53
 LC2, load from editor 45
 lc2gst (see GST_converter)
 LC_OPT (See Environment,
 LC_OPT)

lcompact, header file compressor
 251

LIB (See Environment, LIB)

LIB environment variable 114

Librarian 253

Lattice 253

LC 72

Libraries 121

GEM, linking 66

maths, linking 66

libraries, linker 115

Library files, locating 57

Library map 66

Library path (See Environment,
 LIB)

Line Editing, CLI 138

line length, linker 118

line numbers, debugger (see
 debugger, line numbers)

Line, goto, Editor 28

Linker 2, 113

 BSSBAS 120

 BSSLEN 120

 DATABAS 120

 DATALEN 120

 edata 121

 end 121

 etext 121

 LinkerDB 120

 MERGED 115

 RESBASE, _RESLEN 120

 adding linker symbols 116

 CLink 113

 columns, number of 118

 compiling and linking from Editor 37

 cross referencing 117

 debug suppression 116

 Errors 123

 extended format 116

 file name field width 118

 from files 114

 hunk field width 118

 keywords

 ADDSYM 116

 FROM 114

 FWIDTH 118

 HEIGHT 118

 HWIDTH 118

 INDENT 118

 LIB 115

 LIBRARY 115

MAP 117

ND 116

NODEBUG 116

PRELINK 116

PWIDTH 118

SWIDTH 118

TO 116

WIDTH 118

XADDSYM 116

XREF 117

LC 66, 72

adding symbols, -La 66

automatic program detection 72

batch mode, -Lb 66

cross reference, -Lx 67

desk accessory 72

GEM, -Lg 66

hunk map, -Lh 66

library map, -Ll 66

map file, -Lf 66

maths, -Lm 66

removing debugging, -Ln 66

verbose mode, -Lv 67

libraries 115

library ordering 121

line length 118

linking from Editor 37

listing indent 118

map file 117

number of columns 118

output filename 116

pre-linking 116

program width 118

source files 114

symbol width 118

symbols 116

symbols, Editor 40

width, file name field 118

WITH file examples 118

Linking with GEM 66

Linking with maths 66

LinkST (see GST, linker)

LIST directive, assembler 221

listing

 disable, assembler, NOLIST 221

 indent, linker 118

 page throw, assembler, PAGE 222

 listing, source (See source listing)

 LNG file format 302

 Load automatically 49

 load binary, debugger 194

 load global data register 73, 79

 load program, debugger 194

load source, debugger 195
 loading
 resource editor 156
 Loading text, Editor 31
 Locating
 header files 56, 65
 intermediate files 57
 library files 57
 programs 56
 Locating include files 65
 long alignment 67
 longword align data 61
 Low Resolution
 debugger (see Debugger, low
 resolution)
 resource editor 145
 L.PTR pre-processor symbol 59

M

M68000 pre-processor symbol 58
 macro arguments, assembler, number
 of, NARG 221
 Macro Assembler (See Assembler)
 macro definition, assembler,
 MACRO 221
 MACRO directive, assembler 221
 macros
 expansion listing 65, 217
 make option 68
 mantissa 266, 267
 map

cross reference 67
 hunk 66
 library 66
 map file
 linker 117
 map file, generating 66
 maths co-processor 63
 maths, linking 66
 maximum errors 70
 maximum warnings 70
 memory layout, debugger 207
 Memory, 512K against 1Mb+ 4
 Menu, resource editor 167
 MEXIT directive, assembler 221
 mixed precision, force 64
 MKDIR, CLI 135
 modifiers
 const 75

signed 76
 volatile 77
 Modula-2, resource editor 296
 MonST2C (see debugger)
 MOUSE, CLI 135
 MSDOS 129
 multiple character constants, -cm 61
 multiple include suppression 61

N

naming of libraries 121
 naming, resource editor 162, 166
 NaN 266, 267
 NARG, assembler reserved symbol
 221
 narrow listing 65
 near keyword 59, 71, 78
 Neochrome 260
 new keywords, permit 61
 New, resource editor 156
 No more errors 37
 NOLIST directive, assembler 221
 non-base-relative 59, 77, 119, 122
 Not-A-Number 266, 267
 number of columns, linker 118
 Number representations, assembler
 212
 number select, resource editor 164
 numbers, debugger 183

O

ob_head 284
 ob_next 284
 ob_tail 284
 object file, placing 69, 216
 object module disassembler, omd 252
 object module librarian, oml 253
 object, struct definition 282
 object-level editing, resource editor
 146
 objects 141
 Copying 161
 Flags (See Resource Editor:flags)
 Resource Editor 143, (see Resource
 editor, objects)
 Octal, assembler 212
 OFFSET directive, assembler 221
 omd, object module disassembler 252

oml, object module librarian 253
 opaque, resource editor 160
 opcode, assembler 212
 operand, assembler 212
 operation, assembler 212
 operators
 assembler 213
 debugger 182
 optimisation
 deferred stack cleanup 68
 registerisation 68
 space 68
 time 69
 Options
 \$ legality mode, -cd 61
 68000 family mode, -ma 68
 68000 mode, -m0 68
 68010 mode, -m1 68
 68020 mode, -m2 68
 68030 mode, -m3 68
 absolute code, -r0 71
 adding debugging, -Ln 66
 adding symbols, -La 66
 align long objects, -l 67
 allow register keywords, -c 61
 ANSI compatibility mode, -ca 60
 Auto-detecting 68881 mode, -fa 63
 automatic registerisation, -m 68
 base-relative, -b1 59
 batch mode, -Lb 66
 big compiler, -B 59
 C++ mode, -c+ 60
 char unsigned, -cu 62
 code generation, -m 68
 code model, -r 71
 compatibility mode, -c 60
 Compiler 39, 59
 constant string copying, -cs 61
 continuous compilation, -C 60
 control errors/warnings, -j 65
 cross reference compiler includes, -gc
 64
 cross reference defines, -gd 64
 cross reference linker symbols, -Lx 67
 cross reference symbols, -gx 65
 cross referencing, -g 64
 data model, -b 59
 debugging mode, -d 62, 176
 default 57
 deferred stack cleanup, -mc 68
 define symbols, -d 63
 disable debugging, -d0 62
 dual standard, -rb 71
 eliminate static prototypes, -pe 70
 enable structure warnings, -ct 62
 enforce prototyping, -cf 61
 environment, -E 56, 63
 error source line suppression, -ce 61
 excluded source listing, -ge 64
 extended character set, -e 63
 extract portable prototypes, -pp 70
 extract prototypes, -pr 70
 extract static prototypes, -ps 70
 floating point, -f 63
 force double mode, -fd 64
 force mixed mode, -fm 64
 force single mode, -fs 64
 full debugging 62
 generate GST code, -z 73
 generate precompiled header, -ph 69
 hunk map, -Lh 66
 I/O mapped 68881 mode, -fi 63
 identifier significance, -n 69
 invoke global optimiser, -O 69
 Lattice IEEE mode, -fl 64
 LC_OPT 57
 librarian, -R 72
 library map, -Ll 66
 line debugging, -d1 62
 linker verbose mode, -Lv 67
 linking automatic detection, -td 72
 linking desk accessory, -ta 72
 linking with GEM, -Lg 66
 linking with maths, -Lm 66
 linking, -L 66
 list compiler includes, -gh 64
 list macro expansions, -gm 65
 list user includes, -gi 64
 load global data register, -y 73
 load pre-compiled header, -H 65
 longword align data, -cl 61
 make char unsigned, -cu 62
 make globals external, -cx 62
 make globals external, -x 73
 make option, -M 68
 map file, -Lf 66
 maximum errors/warnings, -j 70
 Motorola 68881 mode, -f8 63
 multiple character constants, -cm 61

narrow listing, -gn 65
 non-base-relative, -b0 59
 object file, placing, -o 69
 PC-relative code, -r1 71
 permit new keywords, -ck 61
 pre-ANSI pre-processor, -co 61
 pre-processor only, -p 69
 pre-processor, -p 69
 promote warning to error, -jne 65
 QUAD file, placing, -q 70
 real maths, -f 63
 registerised entry, -rr 71
 return value warnings, -cw 62
 section naming, -s 72
 set environment, -E 56, 63
 set INCLUDE path, -i 65
 short integer, -w 73
 space optimisation, -ms 68
 stack checking, -v 73
 standard stack, -rs 71
 startup code, -t 72
 strengthen aggregate equivalence, -cq 61
 suppress multiple includes, -ci 61
 suppress source listing, -Gs 65
 suppress warning, -jni 65
 time optimisation, -mt 69
 undefine symbols, -d 73
 unsigned char, -cu 62
 Options menu, Editor 39
 output filename linker 116

■ P

PACKING bit-fields 269
 PAGE directive, assembler 222
 Page down, Editor 27
 page throw, assembler, PAGE 222
 Page up, Editor 27
 parameter passing, register 61
 parent, resource editor 148, 160
 PARMBLK 291
 PARMBLK structure 290
 Parser 52
 Paste block, Editor 35
 paste, resource editor 161
 PATH Environment 56, 137
 Saved! 50

PAUSE, CLI 136
 PC-relative code 71, 119, 228
 portable prototypes 70
 pre-ANSI pre-processor, -co 61
 Pre-compiled header 61, 74
 building 74
 generate 69
 load 65
 using 74
 pre-defined symbols 58
 pre-linking, linker 116
 Pre-processor 52, 58
 DEBUG symbol 62
 define symbols 63
 options 69
 eliminate static prototypes, -pe 70
 extract portable prototypes, -pp 70
 extract prototypes, -pr 70
 extract static prototypes, -ps 70
 generate precompiled header, -ph 69
 pre-process only, -p 69
 pre-ANSI 61
 pre-defined symbols 58
 quoted string substitution 61
 suppress multiple includes 61
 undefine symbols 73
 precedence, operator, assembler 213
 precision double 64
 mixed 64
 single 64
 prefacing, underline 217
 Preferences debugger (see Debugger, preferences)
 Editor 43
 resource editor 157
 prefix, resource editor 162, 167
 Print block, Editor 35
 Printer, the Install desk accessory 35
 processor target 217
 ProgDef, resource editor 144
 program width, linker 118
 programming with resources 282
 Programs, locating 56
 prototypes eliminate static 70
 enforce 61
 extract all 70
 extract static 70
 portable 70

■ **Q**
 QUAD (See Environment, QUAD)
 QUAD files (See intermediate files)
 Quit, Editor 33

■ R

Radio Button, flag type 286
 ramdisk 249
 real maths (See floating point)
 Redirection, CLI 140
 register keywords, allow 61
 register passing 122
 registers, debugger 184
 Registration card 1
 relocatable expression, assembler 213
 relocatable origin, assembler, RORG directive 222
 REM, CLI 136
 removing debugging 66, 258
 removing symbols 258
 REN, CLI 136
 Replace all, Editor 30
 Replacing, Editor 29
 reserved symbols linker 120
 resident program 333
 Resource editor 3, 141
 Abandon Edit 162
 alert 169
 alphabetic 165
 APPLBLK 290
 assembly language 294
 auto naming 149, 162
 Auto Size 162
 auto snap 152, 162
 BASIC 294
 BITBLK 289
 border colour 160
 menu 160
 thickness 287
 box 144, 273, 284
 BoxChar 144, 150, 273, 285
 BoxText 144, 273, 285
 Button 144, 273, 285

C 295
 Cancel 161
 Checked 279
 child 284
 child number 159
 Clipboard 161
 colour border 160
 fill 160
 names 288
 numbers 288
 word 287
 control characters 150
 Copy 161
 copying trees 167
 Crossed 279
 Cut 161
 Default 277
 Delete 162
 deleting trees 167
 Disabled 279
 Editable 278
 editing icons 154
 editing images 152
 example program 297
 Exit 278
 Expert level 165
 extended type 159
 extras 152, 159
 FBoxText 144, 150, 274, 286
 fill colour 160
 Fill Menu 160
 Find name 163
 text 163
 Flag Types 277
 flags 143, 158
 menu, 158
 summary 280
 flags menu 158
 Forms 166
 FORTRAN 295
 free image 171
 string 168
 FText 144, 150, 276, 285
 grand child 284
 greyed. 279
 Half Char Snap 163
 half character snap 152, 163
 head 142

cross reference, linker 67
 debugger 184
 linker 116
 pre-defined 58
 symbols, removing 66, 258
 Syntax check, Editor 36

■ **T**
 Tab key, Editor 27
 target processor 217
 Technical support 1, 347
 TEDINFO
 resource editor 149, 150
 TEDINFO structure 288
 temporary equate, assembler, SET 222
 terminate, debugger 194
 test, resource editor 165
 Text
 clear, Editor 30
 delete, Editor 32
 deleting, Editor 30
 insert, Editor 32
 load, Editor 31
 resource editor (see Resource Editor, text)
 save, Editor 31
 Thorough check, Editor 36
 time optimisation 69
 Title, resource editor 144, 168
 Tools 249
 Tools menu (see Editor, Tools)
 Top of file, goto, Editor 28
 TOS
 changing fonts, Editor 41
 setting command line from Editor 38
 tool type, Editor 46
 Touch Exit, flag type 286
 transparent, resource editor 160
 TRAPV instruction 206
 Tree level editing
 resource editor 146, 166
 tree level editing, resource editor 147
 tree name box
 resource editor 148, 166
 trees, resource editor (see Resource Editor, trees)
 TTL directive, assembler 222

TTP
 setting command line from Editor 38
 Tutorial 4
 compiler errors 11
 compiler options 18
 compiling and linking 6, 9
 using GEM 16
 TYPE, CLI 137
 types
 bit-field 269
 enum 75
 void 76

■ **U**
 undefine pre-processor symbols 73
 undefined structure warnings 62
 Undelete Line, Editor 30
 underline prefacing 217
 underline, resource editor 150
 Undo line delete, Editor 30
 union children, resource editor 158
 union, alignment 269
 unsigned char 59, 62
 Upgrades 347
 user includes, listing 64

■ **V**
 Valid, resource editor 150
 variable, assembler 212
 variable, program counter, assembler 212
 verbose mode 67
 VIRTUALDISK, CLI 138
 void type 76
 volatile modifier 77
 VT52 screen codes 339

■ **W**
 Warnings 84
 control 65
 enabling 66
 function return value 62
 maximum 70
 promotion to error 65
 suppressings 65
 undefined structure tag 62
 wconvert 259

WERCS 3, (See Resource Editor)
 WERCS.INF, resource editor 157
 What blocks! 34
 What errors! 37
 WHICH, CLI 138
 width, file name field in linker 118
 Width, resource editor 159
 winage 260
 Windows
 arrange in Editor 40
 cut & paste in Editor 48
 cycle in Editor 41
 debugger (see Debugger, windows)
 switching Editor windows 48
 with file
 linker
 examples 118
 Word
 left, Editor 27
 right, Editor 27
 WordStar keys, Editor 27
 writing assembly language 225
 WTEST 297

■ **X**

XDEF directive, assembler 222
 XREF directive, assembler 222

■ **Z**

zero divide 206
 zoom window, debugger 189