The relocation type and size parameters of the CSECT directive have been used in order to provide automatic PC and A4 base-relative addressing modes for the relevant sections. This has the effect of automatically converting the references to string and hash to:

```
lea     string(a4),a0
jsr     hash(pc)
```

Simply changing the relocation type allows the assembler to automatically generate the correct addressing modes. This corresponds directly to the C compiler options. To override the default addressing mode you may simply specify another, or for external symbols, provide an XREF in an appropriate control section. In our example, moving the reference to @hash outside the PC-relative section forces absolute long addressing for all references to that symbol.

Specifying the special section name of __MERGED causes the linker to include the section contents within the program's near data segment allowing base-relative addressing via A4.

Now the hash function written in C:

```
/* declaration */
extern unsigned short maxhash;

unsigned int __regargs
hash(unsigned int length, const char *string)
{
        unsigned long total = 0;

        while (length--)
                total += *string++;

        return total % maxhash;
}
```

The __regargs keyword is present to force the compiler to use register passing for this function. Remember to link with the startup code and libraries for register parameters since we are using @main rather than _main.

Asm Error Messages

Branch out of range for 8-Bit offset

A short branch to a label outside the range of an 8-bit offset was specified. This can be cured by simply changing the branch size to word.

Branch out of range for 16-Bit offset

A word branch to a label outside the 16-bit range was specified.

Can't branch short to EXTERN

Asm does not allow short branches to an external label, causing this error.

Can't create object file

It was not possible to generate the object file. This can be caused by invalid -o options, disk full, protected files, etc.

Can't open source file

The source file could not be opened, often caused by an incorrect filename.

Combined output file name too large

The object file prefix specified from the command line caused the output filename to exceed the maximum length of 128 characters.

Constant size not same as relocation size

A reference to a relative symbol conflicts with the byte size of relocation information for the current control section.

Constant too large

An invalid ROM constant number was specified for an FMOVECR instruction.

Data generation must occur in reloc section

A data generation operation other than DS appeared in an OFFSET section.

Definition symbol not found

Internal error, should never happen.

Duplicate label

More than one definition of the same label was encountered.

Duplicate macro definition

A macro was defined more than once.

Duplicate section name

A section name was re-used illegally.

ELSE/ENDIF not found

An unterminated IFcc directive was encountered.

END directive assumed (W)

This warning notifies you that there was no explicit END directive in the source file being assembled.

Error writing object file
Execution terminated

A DOS error occurred whilst writing the object file to disk.

Errors detected -- Processing terminated

This message appears at the end of the assembly process if any errors occurred. Any object file generated will be invalid.

External name table overflow

The maximum number of imported labels exceeded the maximum of 256.

External symbol defined

A definition for a label also declared as an external reference was encountered.

Extraneous data on input line (W)

A valid source line was followed by invalid text, which was ignored. This can be caused by providing too many parameters for an assembler directive.

File name missing

The command line did not contain a file to assemble.

File not found

The file specified by an INCLUDE directive could not be found.

Generating 32 bit branch, code only valid on 68020

This warning is generated when assembling a long branch for the 68000 processor.

Generation argument count

Internal error, should never happen.

Illegal macro definition

The syntax of a macro definition was incorrect.

Immediate data size error
Immediate data too large

An arithmetic or logical operation was specified with an out of range immediate value.

Immediate mode not allowed

An instruction which does not support immediate addressing was encountered with an immediate mode operand.

Input line too large

A source line exceeded the maximum length of 255 characters.

Invalid Addressing Mode

Generated by an illegal addressing mode being supplied to certain 68020 or floating point instructions.

Invalid command line option

The assembler was invoked with an unrecognised command line option.

Invalid control section parameter

A CSECT directive with invalid parameters was encountered.

Invalid Data Size

The vector of a BKPT instruction was out of range.

Invalid destination mode

The second operand of an instruction was specified with an illegal addressing mode.

Invalid Effective Address for Opcode

One of the address translation cache family of instructions contained an illegal addressing mode.

Invalid expression

An OFFSET or IF directive contained an invalid expression. This error can also be caused by an expression containing a divide or modulo by zero.

Invalid file name

The filename specified for an INCBIN or INCLUDE directive was not valid.

Invalid generation function index

The assembler attempted to generate an illegal instruction.

Invalid Length

A LINK instruction was encountered with an illegal stack offset.

Invalid list option

The assembler command line contained an unsupported listing option.

Invalid mode

An illegal addressing mode was used with an instruction.

Invalid opcode

An unrecognised opcode name was encountered; this is often caused by a mis-typed instruction.

Invalid operands for this opcode

This error can be caused by invalid addressing modes, data size, macro parameters etc.

Invalid origin

An assembly directive causing incorrect data alignment or origin was found.

Invalid relocation type/size combination

The specified relocation type and relocation data size specified in a CSECT directive are not available.

Invalid shift count

The bit count contained in a shift or rotate instruction was out of range.

Invalid Size
Invalid Size Field
Invalid Size For Mode

Each of these errors are caused by an invalid or unsupported data size extension to an instruction or addressing mode.

Invalid source

A MOVES instruction was specified with an invalid source operand.

Invalid source mode

The first operand of an instruction contained an unsupported effective address.

Invalid string

A define constant or condition directive contained an invalid string.

Invalid symbol

A symbol containing an illegal character or characters was declared.

Invalid vector

This error is generated if a TRAP instruction has an out of range vector.

-i option ignored

The maximum number of include directories which can be specified on the command line has been exceeded.

k-factor out of range

The k-factor specified for an FMOVE instruction on packed data was out of range of the legal values.

Label ignored (W)

The label before a directive, such as a conditional, is not a recognised syntax and has been ignored.

Label not found in pass 2

Internal error, should never happen.

Label offset different in pass 2

A phasing error caused by different code being generated on the first and second pass.

Lexical result overflow
Lexical type error
Lexical value overflow

Internal errors, should never happen.

Long Branches to EXTERNs not supported by the Linker

The Lattice linkable object file format does not support branches to external labels using a long-word offset.

Macro argument too large

A macro invocation was encountered with an argument string which was too long.

Macro buffer overflow

A macro definition was too long.

Macro nesting level exceeded

This error occurs when a macro definition references other macros too many levels deep.

Macro substitution line overflow

The substitution of macro arguments caused the line to overflow.

Maximum include file nesting exceeded

The INCLUDE directive has nested files too deeply. This is caused by included files referencing other files to a number of levels.

Missing label

An EQU or SET directive was encountered with no corresponding label.

Missing macro definition

The definition of a macro could not be found.

Must occur inside section

A data generation directive was used outside a control section.

Not enough memory

The assembler ran out of memory when trying to allocate some internal buffer space.

Not inside macro definition

An assembly directive only valid within a macro definition, such as ENDM, was encountered outside a macro.

Not inside scope of IF directive

An ELSE directive was found which did not lie within a conditional control block.

Number Too Big

A value in an expression overflowed the allowable range.

Options beyond file name ignored

Any command line options specified *after*, rather than *before* the file to be assembled have been ignored.

Public symbol not defined (W)

The program source contained an XDEF directive of a symbol which was not defined in the program.

Seek error on object file Execution terminated

An attempt was made to move past the end of an object file. This is usually caused by an invalid RORG directive.

Target out of range

This error is generated if a DBRA instruction to a label which is out of range is found.

Too many control sections

This error signifies that the number of SECTION or CSECT directives has caused more than the allowed number of sections to be generated.

Unknown segment type

The type specifier for a SECTION or CSECT directive was other than CODE, DATA or BSS.

Unrecognized opcode

An operation was encountered which was not recognised as a valid opcode, synonym or macro.

Value out of range for mode

An out of range value was used in an addressing mode.

Value out of range for PC Relative addressing

An out of range value was used in a PC relative addressing mode.

Warning 68020 or 68030 opcode generated (W)

This warning is generated when a non-68000 processor instruction is encountered and can be disabled from the command line by specifying an alternate processor.

Internal Errors

These are internal errors generated when the assembler encounters a situation which should not occur internally. If you encounter one of these please send us an example piece of source code which demonstrates the problem.

Memory freed incorrectly

Mode lexical pointer error

Section not found in pass 2

The Lattice C 5 Tools

Lattice C 5 comes with several tools which are non-essential to the operation of the compiler, but which can enrich the programming environment.

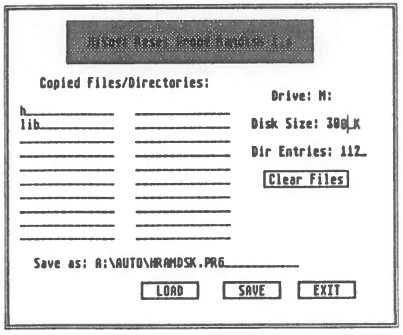
hramdsk — Reset Proof RAM Disk

If you have a megabyte or more of RAM then you can use some of this memory as a very fast disk which will make a hard disk seem slow. The problem with many ramdisks is that they disappear when you press reset. If you are developing a new GEM-based program and are having problems with your mouse or menu you can need to reset quite often.

RAMINST.PRG and RAMINST.RSC let you set up a ramdisk that will survive resets. Whilst this will work in 99% of cases ,occasionally, if a program crashes in a particularly nasty way you will lose the contents of the ramdisk. To avoid this save the source code on to a real disk before running your program.Hramdsk will additionally copy the files and folders that you would like on the ramdisk automatically when you switch on.

The version of HRAMDSK.PRG that we supply with Lattice C is configured for our recommend setup for users with one floppy disk and one megabyte of RAM as described in the Installation Guide. Before running RAMINST to tailor the ramdisk to your preferences, HRAMDSK.PRG should normally be in the AUTO folder on the current disk as this will be used as a basis for the ramdisk driver.

To run RAMINST copy it just double-click on RAMINST.PRG. If the driver cannot be found in the AUTO folder is not you will be presented with a file selector to enter the drive, path and file to load. Once the driver has been loaded, RAMINST will present you with a GEM dialog box like this:



You can change the files and folders that are copied by clicking on one of the relevant fields and editing it. To clear out the existing names click on the Clear Files button.

You can use simple filenames, filenames with wildcards or directory names. If you specify a directory name the entire directory (and any sub-directories within it) will be moved to the ramdisk. Note that you can specify the drive from which the files are copied.

Change the size of the ramdisk by simply modifying the Disk Size field. This is the amount of memory used by the disk and is slightly larger than the data size because some of the space is used for the directory and the file allocation table. You can use any size you like, subject to the available amount of RAM.

Normally you can have up to 112 directory entries in your main directory (just like a standard floppy disk) and this is usually sufficient. However you may change the maximum number of directory entries using the Dir entries field of the dialog box. For example, if you have a two and a half Megabyte ramdisk and using it to store (amongst other things) all the relocatable files for your 100 module wonder program and don't want to put them in a sub-directory, you can use this option to increase the number of directory entries.

If you don't like the ramdisk being called drive M you can change this too. You can also change the disk, directory and file to which the driver will be saved.

Finally if you wish to modify a differently installed version of the ramdisk you can click on the Load button and load another file.

Note that the ramdisk is only designed to be run from the AUTO folder; it can't be run once the system has booted.

lcompact

Header file compressor

This command is used to create a compressed version of a header file that may be processed more quickly by the compiler.

```
lcompact infile outfile
```

The lcompact command compresses a file by performing four basic operations:

Compression

Multiple and meaningless blank characters are removed. Expressions are analysed according to standard C precedence rules to determine which are safe to remove.

Elimination

All comments and unnecessary blank lines are removed.

Transformation

Certain sequences such as hex constants are converted into more efficient representations. For example, 0x0000001 is transformed to 1. These transformations take place only if the secondary representation is more space efficient.

Tokenisation

A limited number of common keywords are reduced to a single token character with the high order bit set. This fixed set of keywords is known to the compiler and will be automatically expanded as they are encountered.

The end result of compacting a header file is anywhere between 20% and 75% smaller depending upon the original contents. The compiler will automatically expand the header file on input so that error messages will print out the original line (without comments) for a diagnostic.

omd — Object Module Disassembler

This utility program disassembles an object file produced by the Lattice C Compiler and produces an output listing consisting of assembly-language statements, possibly interspersed with the original C source code.

```
omd >output options object source
```

The object field is required and gives the complete object file name. That is, omd will not automatically supply the .o extension.

The source field is optional; if present, it must be the complete source file name. When this field is used, you should have compiled the source file with the -d option (see the lc command) to produce the debugging information in the object module that allows omd to associate a particular source line with the object code that was generated. If you did not use the -d option, then the C source lines will not appear in omd's output.

The >output field is optional and redirects omd's output from the screen to an output device or file. Most programmers use omd by redirecting its output to a file and then printing the file. See the example below.

The options field need not be present. The Atari implementation of omd only accepts the following option:

-x Override the default size of the buffer used to hold the external symbol section of the object module. For example, -x200 establishes a buffer that can hold 200 external symbols, which is the default. You should increase this value if omd reports that there are too many external symbols.

Example

This example compiles MYPROG.C to produce MYPROG.O, which is then disassembled. The disassembled listing is saved in the file MYPROG.LST.

```
lc -d myprog
omd >myprog.lst myprog.o myprog.c
```


oml — Object Module Librarian

The object module librarian oml can create a library file by combining object modules, generate a listing of the modules (and their public symbols) contained in a library, or manipulate modules within an existing library file.

Library files provide a convenient way of collecting object modules to be presented as a group of available components during linking; the linker then includes only those modules from the library which are actually needed by the program being built. Libraries are especially useful when several programs make use of common subroutines, since these subroutines can be placed in a library file and included, as required, when the programs are linked.

A library file is made up of object modules, each of which was originally a single file. Each module within the library is identified by a module name, which is normally obtained from the object module itself (the Lattice object module format defines a special *program unit* or module name record). This name is placed in the object module by the translator (assembler or compiler) program which generates it. Some modules may not contain a module name at all; in that case, the librarian assigns a module name of the form $nnn, where nnn is a decimal number indicating that the module was the nth un-named module encountered in the library.

In order to perform replacement of modules within a library file, it is necessary to ensure that the module contains a program unit or module name. The Lattice C 5 Compiler and assembler always place a module name in the object files they produce. The current versions of the compiler and assembler use the name of the object file. Thus, compiling ftoc.c produces an object module with the name ftoc.o.

When the linker examines a library file to find modules to be incorporated into a program, the module name is not important; instead, the linker decides if a module is needed on the basis of the public symbols it defines. A module may define one or more such symbols, which identify program components such as functions or variables. Because the presence of more than one definition for a symbol may cause confusion, the librarian warns when it examines or constructs a library file which includes multiple definitions of a symbol.

Each invocation of oml specifies a particular library file upon which operations are to be performed. Then, a sequence of one or more commands is used to indicate the desired operations.

Commands may be specified on the command line used to execute oml, or they may be read from stdin, or a combination of both. If no commands are specified on the command line, commands are automatically read from stdin, which is usually the user's console but can be redirected to a file. The special command @ (valid only on the command line) is used to switch command input from the command line to stdin; an explicit file name may be attached to the @ to force commands to be read from that file. Commands are read from a file or from stdin until an end of file condition is detected; if commands are being read from the user's console, a Ctrl-Z must be used to end command input.

Each command is specified as a single character, usually followed by a list of module names or object file names. Commands and file/module names are separated from each other by white space. If a command is followed by one or more names, the first name specified is *not* checked as a possible command; thus, names which might be confused with commands must be specified as the *first* name following a command.

The format of the command to invoke the librarian is:

```
oml [<cmdfile] [>listfile] [options] libfile [commands]
```

The various command line specifiers are shown in the order they must appear in the command. Optional specifiers are shown enclosed in brackets.

<cmdfile Causes commands to be read from the named file, provided that no commands are specified on the command line or that the @ command (see below) is not used to force commands to be read from stdin. If this option is omitted and neither of the above conditions is met, commands are read from the user's console.

>listfile Causes the listing output generated by oml to be written to the named file. If omitted, listing output is directed to the console.

libfile Specifies the name of the library file to be created or manipulated; this is the only command line field which *must* be present.

Options are specified as a minus sign followed by a string of characters which may not include white space. The options available are:

-b Forces the 'batch' mode of operation; in this mode no user interaction will occur.

-opfx Specifies that the output filenames for the x command are to be formed by prefacing the module name with pfx. Note that if a directory name is to be specified as a prefix, a trailing node separator (a backslash under GEMDOS) must be supplied on the prefix.

-s Causes a listing of the public symbols defined in the module to be included in the listing produced by the l command.

-v Forces the 'verbose' mode of operation; in this mode the librarian prints out its progress to date.

-x Causes a cross reference of symbols to be output in the listing produced by the l command.

Commands are specified by a single character; if alphabetic, either upper or lower case may be used. They are separated from each other and from elements of file or module name lists by any white space. The commands are:

@file Causes the remainder of command input to be read from stdin, or from the named file.

d list Deletes the named modules from the library. Since modules without program unit names are assigned module names by the librarian, it may be necessary to obtain a listing (via the l command) in order to determine the assigned $nnn name for the module which is to be deleted.

l Causes generation of a listing of the modules in the library after all other requested operations have been performed. If the -s option is used on the command line which invokes oml, the listing will include the public symbols defined in each module as well as a list of the module names themselves.

r list Replaces the named object files in the library, or adds them to the library if not already present. Note that replacement of existing modules in a library will work correctly only if the file name is the same as the module name. Note that a is a valid synonym for r.

x list Extracts the named modules from the library, creating files of the same names. Note that if the module name includes a path name, the librarian will attempt to create a file with that name. All files are created in the current directory unless the -o option is used; in that case, each module name is prefixed with the text specified on the -o option. If the special name * is specified in an extraction then oml will extract all files in the library. oml will terminate execution if an attempt to extract a module is unsuccessful. Note that it is an error to specify the same name in both a replacement and an extraction list.

If replacement modules or deletions are specified, a new version of the library file will be built, provided that no errors are detected. This new version is created first as a temporary file; when it has been completely built, the original library file (if it existed) is deleted, and the temporary file renamed. This sequence ensures that the original library file will not be affected if an error is detected.

Modules are always included in a library in topologically-sorted order, so that no backward references occur (except in the case of modules which reference each other, which are retained in the same order in which they are encountered).

Warning messages are produced if a module named in a deletion or extraction list was not found in the library or if a second definition for a public symbol is encountered in one of the modules to be included.

Examples

The following examples illustrate the use of oml. Remember that replacing modules within a library file only works correctly if the module name is the same as the file name.

Building a New Library

Create a list of the file names of the object modules which will make up the library. Then create the library using the following command:

```
oml new.lib r @name.lst
```

where new.lib is the name of the library to be created, and name.lst contains a list of the files to be included in the library. Note that creation of a new library is one occasion where the correspondence between file and module names is not required.

Extracting Modules from a Library

Use the following command to break out all of the modules from a library:

```
oml -o\object\ cfuncs.lib x
```

Note that this command will be successful only if no module names in the library cfuncs.lib contain path names. A file for each of the modules in the library will be created in the directory \object\ in this example.

Deleting Modules from a Library

Use the following command to delete modules from a library:

```
oml cfuncs.lib d tribe.o
```

This example deletes the module tribe.o from the library cfuncs.lib.

Listing the Modules in a Library

Use the following command to obtain a listing of the modules and symbols in a library file:

```
oml -s test.lib l
```

The listing may be saved to a file using I/O redirection:

```
oml >test.lst -s test.lib l
```

strip

Symbol Strip Utility

Strip is a utility for removing the symbol table and any symbolic debugging information, from an executable file.

```
strip file1 [file 2 ...]
```

Any number of files, which should include any extension, may be specified. If a file is not executable it is simply ignored.

Example

This example compiles MYPROG.C to produce MYPROG.O, which is then linked and then has its symbols and debug information removed:

```
lc -d3 -La myprog
strip myprog.ttp
```


wconvert

Resource Name Converter

Wconvert is a utility for converting the name files from the Digital Research and Kuma Resource Construction Sets. It is provided so that you can edit resource files produced using these programs with WERCS while retaining your names for trees and objects.

There are two versions of this program:

wconvert.prg lets you select the file to convert using the file selector. After it has converted one file it will let you select another one if you wish.

The second version is wconvert.ttp which takes the names of the file(s) to convert on the command line for CLI users. Wildcards may be used with this version.

Wconvert treats files differently depending on their extension:

DEF it is assumed to be a Digital Research RCS1 file.

RSD it is assumed to be a Kuma K-RSC file (actually the same basic format as RCS1).

DFN it is assumed to be a Digital Research RCS 2 file.

The file will be converted into a .HRD file of the same name and in the same directory as the old file, ready for use with WERCS. Remember to make sure that the Language and Case settings are correct when you edit the file with WERCS for the first time.

wimage

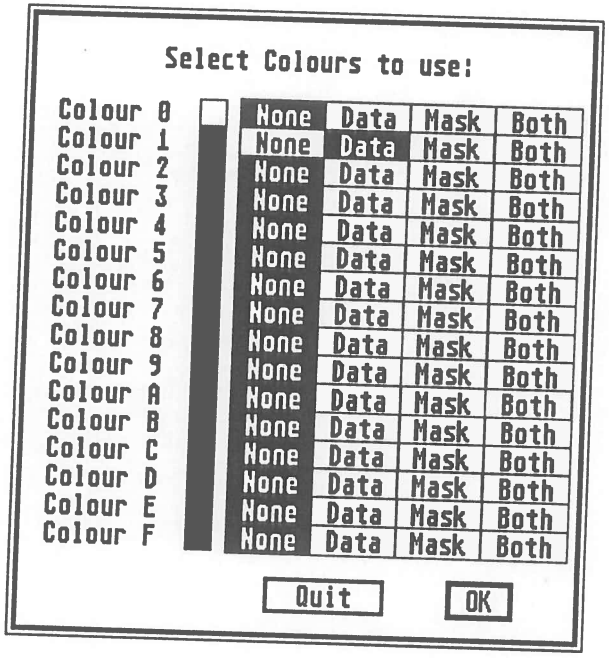
Image Converter

Wimage is a utility for converting parts of Neochrome and DEGAS format files into resource files.

After starting wimage and you will be prompted to enter a file to convert via a File Selector. This file may be a Neochrome format file (normally with an extension of .NEO) or un-compressed Degas/Degas Elite file (normally .PI3, .PI2 or .PI1). Wimage knows about medium and high resolution Neochrome format files even though Neochrome itself does not.

Converting colour pictures to Images and Icons has the disadvantage that GEM Icon and Images have only two colours. Also note that the maximum size of Images and Icons that can be converted is 128x128 pixels.

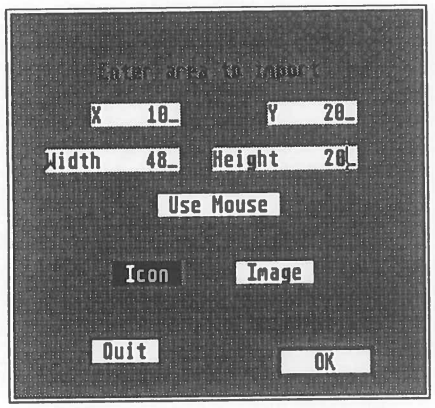
After entering the file name, the Image file will be loaded and you will be presented with a dialog box like this:



The box above is for a low resolution picture. If the picture is a medium or high resolution picture then only the appropriate colours will be displayed. If you are converting from a low resolution picture and the screen is in low resolution mode then the boxes after Colour 2 etc. will be in the appropriate colours as displayed by Neochrome.

This dialog box enables you to indicate which colours are to be treated as Data and Mask bits in the Icon or Image that wimage produces. If Both is selected for a particular colour then the corresponding pixels of this colour will be set in *both* the Data and Mask. If Data is selected they will be *set* in the Data but *reset* in the Mask. If Mask is selected then pixels of this colour will be *set* in the Mask but *reset* in the Data. If None is chosen then the bits will be reset in *both* the Data and the Mask. When importing an Image, only the Data setting is used.

You will then be presented with a dialog box like this:



Enter the area of the picture that will be converted. Indicate whether you are producing an Icon or an Image by clicking on the appropriate radio button.

If you wish to use the mouse to select the area, ensure that the Use Mouse button is selected; this button will be disabled if this picture may not be displayed in the current resolution. If Use Mouse has not been selected then the area to import is taken from the co-ordinates entered.

If you click on OK you will then be prompted to use a File Selector to enter the output file to which your Image or Icon will be written in the form of a resource file. This can be imported into another resource file using the Import Image item from the WERCS File menu.

After the file has been saved you will be given the opportunity to import another image or to quit the Wimage program.

Appendix A Implementation Behaviour

This appendix tries to detail the areas Lattice C 5 which are traditionally different across compilers, and often left undefined. Reliance on any of this information will, in general produce non-portable programs; note that this *includes* reduced portability to other Lattice C systems.

This section is based around Appendix F.3 of the ANSI C Standard document and the paragraph numbers relate to those paragraphs in that standard.

Translation

2.1.1.3

Diagnostic messages are issued on the console device describing the error situation encountered. For syntax violations the compilation is subsequently terminated.

Environment

2.1.2.2.1

The arguments to main() are parsed from the command line as passed to the program via the GEMDOS Pexec() call. Whitespace characters are considered to be separators unless enclosed in single or double quotes. The argv(0) string points to an empty string as the program name is not available.

When started by a process supporting the Atari extended command line format, the arguments to main() are as supplied by the parent process.

A third argument is passed to main() representing the environment variable vector, as described under environ.

2.1.2.3

An interactive device is assumed to be those for which GEMDOS indicates that Fseek() cannot be performed.

Identifiers

3.1.2

The number of significant characters is specified by the -n compile time option. This has a default of 31, and a maximum of 100.

The linker treats all characters as significant, with full case significance.

Characters

2.2.1

The source and execution character sets are the Atari ASCII set.

2.2.1.2

No shift states are used for encoding of multibyte characters.

2.2.4.2.1

There are always eight bits in a char giving a range from -128 to 127 for signed characters or 0 to 255 for unsigned.

3.1.3.4

The source character set is mapped as-is to the execution character set

Multiple character constants are supported when using the -cm flag. The lexical characters are parsed right to left and are packed low byte upwards. Hence the character constant 'ALEX' would pack:

31-24	23-16	15-8	7-0
'A'	'L'	'E'	'X'

The "C" locale is used to convert multibyte characters into corresponding wide characters (codes) for a wide character constant.

3.2.1.1

The type char defaults to signed, but may be configured as unsigned at compile time (-cu).

Integers

3.1.2.5

All integer types are two's complement.

Type int is configurable to be 16 or 32 bits. The default setting is 32 bit, with 16 bit being available when using the -w option.

Type short is 16 bits; type long is 32 bits.

3.2.1.2

The result of converting an integer to a shorter signed integer, or the result of converting an unsigned integer to a signed integer of equal length is truncation to the lower bits of the assigned integer.

3.3

Bitwise operations on signed integers produce results as if the integers were unsigned.

3.3.5

The sign of the remainder on integer division is the same as that of the quotient.

3.3.7

Right shift of a negative-valued signed integral type produces a sign extended type.

Floating Point

3.1.2.5

The floating point format used is the IEEE standard format for both float and double. long double is implemented as double in the current release.

Single-precision Floating Point

The single precision IEEE format represents a number in 4 bytes. Note that all calculation is performed using doubles so that use of floats will only reduce storage space and not increase the speed. The bit layout is:

31	30-24	23-0
Sign	Exponent	Mantissa

The sign bit is 0 for positive numbers and 1 for negative numbers. The mantissa has an implied binary point at bit 23 and thus ranges in value from 1.0 to <2.0. The exponent is held in excess 127 format. The IEEE denormalised format is not currently supported. When the exponent is 255, the value represents Not-A-Number (NaN), the type of which is determined by the mantissa. Zero mantissas indicate Infinity (∞), whilst non-zero mantissas indicate other NaN conditions. The number zero is represented by all bits zero.

The following (non-portable) definition may be used to access the individual fields of a float atomically:

```
union
{
  float f;
  struct
  {
    int s:1;  /* sign */
    int e:8;  /* exponent */
    int f:23; /* mantissa */
  } b;
};
```


Double-precision Floating Point

The double precision IEEE format represents a number in 8 bytes. The bit layout is:

63	62-52	51-0
Sign	Exponent	Mantissa

The sign bit is 0 for positive numbers and 1 for negative numbers. The mantissa has an implied binary point at bit 51 and thus ranges in value from 1.0 to <2.0. The exponent is held in excess 1023 format. The IEEE denormalised format is not currently supported. When the exponent is 2047, the value represents Not-A-Number (NaN), the type of which is determined by the mantissa. Zero mantissas indicate Infinity (∞), whilst non-zero mantissas indicate other NaN conditions. The number zero is represented by all bits zero.

The following (non-portable) definition may be used to access the individual fields of a double atomically:

```
union
{
  double d;
  struct
  {
    int s:1;  /* sign */
    int e:11; /* exponent */
    int f1:20;  /* high bits of mantissa */
    int f2:32;  /* low bits of mantissa */
  } b;
};
```

3.2.1.3

In default mode (-fl) truncation of an integer to floating point is towards zero. In -f8 mode the direction of truncation is as specified by the maths coprocessor (user-selectable).

3.2.1.4

In default mode (-fl) truncation of an floating point number to a narrower floating point type is towards nearest. In -f8 mode the direction of truncation is as specified by the maths coprocessor (user-selectable).

Arrays and Pointers

3.3.3.4, 4.1.1

size_t is *always* of type unsigned long int. Traditionally it has been unsigned int.

3.3.4

Casting a pointer to an integer requires that the target type be long (or int in default long integer mode), or truncation will occur. Casting an integer to a long is as if the integer were first extended to long.

3.3.6, 4.1.1

ptrdiff_t is of type long.

Registers

3.5.1

The compiler honours as many register variable declarations as possible on a lexically first encountered basis for integral, pointer and floating point types (when the -f8 option is used). If the global optimiser is used all registers are remapped to provide maximal usage.

The number of registers available for register declarations is not fixed and hence is undefined. Typically a minimum of 4 scalar and 2 pointer variables will be available.

Structs, Unions, Enums, and Bit-fields

3.3.2.3

Accessing a member of a union object using another member of a different type produces undefined behaviour.

3.5.2.1

The members of a structure are aligned according to the alignment restrictions of the basic type. Hence the structure:

```
struct
{
  char x;
  char y[3];
};
```

would *not* align y. Whereas the structure:

```
struct
{
  char x;
  int y[3];
};
```

would align y. The amount of padding inserted (when required) is determined by the -i flag which will force long word alignment. When -i is not present the default alignment is word alignment.

A plain int bit-field is treated as an unsigned int bit-field

The members of a bit-field are always elements of a long. This means that the members are also of type long.

A bit-field *never* straddles a long boundary.

A bit-field is packed from the top down hence the definition:

```
struct
{
  unsigned abcde:5;
  unsigned fghijk:6;
  unsigned lmn:3
  unsigned opqrs:5;
  unsigned tuvwxyz:7;
};
```

would pack as:

31-27	26-21	20-18	17-13	12-6	5-0
abcde	fghijk	lmn	opqrs	tuvwxyz	??????

3.5.2.2

Enumeration types have the integral type int.

Qualifiers

3.5.3

Whether a reference to an object with volatile qualified type between sequence points constitutes an access is undefined.

Declarators

3.5.4

The maximum number of declarators that may modify an arithmetic, structure, or union type is limited by available memory.

Statements

3.6.4.2

The maximum number of case values in a switch statement is limited by available memory.

Preprocessing Directives

3.8.1

The value of a single-character character constant in a constant expression that controls conditional inclusion matches the value of the same character constant in the execution character set. Such a character constant may have a negative value.

3.8.2

System include files (those in < > brackets) are located initially in the current directory, then in the directory mentioned in the INCLUDE environment variable, then in directories mentioned on the command line via the -I option.

User include files (those in " "s) are located initially in the current, then in directories mentioned on the command line via the -I option, finally in the directory mentioned in the INCLUDE environment variable.

The mapping of includable file names to external source names is obtained by taking the last eight characters of any 'directory' portions (\ or / delimited) together the same portion of the final filename and any extension.

3.8.6

The behaviour on the #pragma inline directive is as described in the section **LC, The Compiler**. This is the only supported #pragma directive.

3.8.8

The definitions for __DATE__ and __TIME__ when respectively, the date and time of translation are not available are the date and time of compiler build.

Appendix B
Resource Details

This appendix contains detailed information on programming with resource files, compiling the WERCS example program and the file formats used by WERCS.

This section of the manual is designed to help you when programming with resource files that have been created by WERCS. The first section describes the objects and their attributes, whilst the second considers the programming aspects of these data structures. It continues by considering the multi-language support available from WERCS and concludes by discussing the WERCS' specific file formats.

Objects

There are thirteen different types of objects as follows:

Box

A Box is a rectangle whose interior colour and fill pattern is controllable, as is its border thickness.

BoxChar

A BoxChar is a graphic box containing a single text character. It also has colour and border size attributes.

BoxText

Similar to Text objects (see later) but in addition surrounded by a border whose size and colour can be specified.

Button

A Button is displayed as a centred string of characters with a border. If the *default* flag is set, pressing the Return key in the standard form-handler is the same as clicking on the Button. A Default button is shown with a wider border. A new Button created with WERCS has its Selectable and Exit flags set.

FBoxText

An FBoxText object is a formatted BoxText object; in addition to the normal Text attributes, it also has border attributes, a template and two extra strings for text entry. These extra strings are called the Template and Valid fields. The AES displays the Template string as if it were displaying any other type of text except that for every underline character it displays a character from the main text string. For example, a date field object might consist of a Text entry of:

```
011088
```

and a Template of

```
Date:__/__/__
```

this would be displayed as

```
Date:01/10/88
```

if it were an FBoxText.

Remember: underline characters are entered as tildes (~).

The Valid string is used when the object is used as a Form using the GEM form_do command. The Valid string specifies which characters may be typed for each underline character in the Template string. The different validation characters are:

X	all characters allowed
9	Only 0-9
A	A-Z and space
a	A-Z, a-z, 128-255 and space
N	A-Z, 0-9 and space
n	A-Z, a-z, 0-9, 128-255 and space
P	A-Z, a-z, 0-9, 128-255, \ : ? * . _
p	A-Z, a-z, 0-9, 128-255, \ : _
F	A-Z, a-z, 0-9, 128-255, : ? * _
f	A-Z, a-z, 0-9, 128-255, _

In the above A-Z includes non-English capital letters. 128-255 means that all characters greater with value greater than 128 can be used, including lower case non-English letters and the £ sign.

You can use different validation characters in the same string if you wish. Thus for the date example above we would use the 9 character for all 6 character positions since the only characters allowed in dates are digits.

The most commonly used of these validation digits are probably X and 9. Note that if you wish to enter negative numbers you have to use X (otherwise the - sign would not be allowed).

Also the pathname options (P and p) are of limited value as a number of software producers sadly use illegal pathname characters such as - in their filenames. All but the X, F and f validation characters also have the undesirable feature with the first, 1.0, operating system ROMs of crashing the system when you press _!

Note that whilst validation characters, P, p, F and f allow you to enter lower case letters these are echoed as the upper case equivalents.

If otherwise illegal characters are present in the Template string then the AES will skip past them if they are entered. With our date example typing / will skip to the next field even though / is not otherwise a valid character.

You should ensure that there are at least as many underlines in the Template string as there are characters in the main Text string; otherwise all of the latter will not be displayed. If you are intending to use this object as a Form *in situ* as normal, you should have the same number of characters in the Text string as there are underlines in the Template; if you don't observe this then if the user types a long string, the next string from your resource file will be corrupted. This restriction does not apply if you are intending to change the address of the Text field when the resource file is loaded.

So, in general, ensure that there are exactly the same number of characters in your Text string as there are underlines in your Template string.

Surprisingly, the Valid string does not have to contain a character for every underline in the Template string; if all the validation characters are the same then you can use just one. We have not seen this officially documented but it certainly works on all versions of the operating system we have used and can lead to considerably reduced resource file and memory usage if you have long strings.

If the first character in a text field is the at-sign (@), then form_do will display your string as underlines and place the cursor at the start of the string. Thus you can enter a blank string of n characters by typing, say ~, n-1 times, press cursor left until you are at the start of the string and then press @. The string will then disappear; but don't worry; it will be stored in your file ready for use.

FText

Similar to a FBoxText (see above) but without border attributes.

IBox

An IBox is a so-called *invisible box*, similar to a Box but hollow. It is only truly invisible if its border has a thickness of zero.

Icon

This consists of two bit-map images, one for data and one for a mask. In addition a string of characters and a single character are also associated with it. Icons also have their own foreground and background colours.

Image

An Image is a graphic bit-map with a foreground colour attribute. It differs considerably from an Icon; it has no mask (so cannot be distinguished on a patterned background), and no associated text or single character.

ProgDef

This type of object is for advanced programmers only. It allows you to create your own types of object by supplying your own drawing routines. ProgDefs are displayed in WERCS as boxes with a diagonal line. ProgDefs are also known as *UserDefs*. We have seen the latter term used mainly in older documentation; a hangover perhaps from days when a Digital Research *user* was someone who wrote the assembly language to install CP/M on their computer.

String

A String is a sequence of characters drawn in black and in the standard system font. If you require different sized or coloured text you should use one of the formatted text object types.

Text

Actually graphic text; this is a sequence of characters that can be displayed in the system font or in a small font and can be left-, centre- or right-justified.

Title

Objects of type Title are only used as Menu titles. Their use in other types of tree is not recommended; they have the same attributes as Strings.

Flag Types

The different flag types for objects are as follows:

Selectable

The Selectable flag is used in conjunction with the form_do AES routine. If this flag is set then if the user clicks on the object during a form_do call it will be highlighted and the Selected state bit will be set. If the Selected bit was already set, the object will be shown as normal and the Selected bit reset.

Thus setting this bit effectively turns any object into a Button without changing the appearance of the object. All Buttons that are to be used as such should have this bit set; this is the default when you create a new Button with WERCS.

Default

The Default bit tells the AES that this is the default button of the form, i.e. the one which will be returned if the user types Return.

Normally this is used for Buttons but can also be used for other types. With Buttons the Default bit causes the object to be displayed with a wider border so that the user can see the default. For other objects there is no change in the screen display.

If the Default bit is set for an object you should normally also set the Selectable and Exit bits.

We do not recommend having more than one Default item in a form; the user, your program and the AES are likely to get confused.

Exit

The Exit bit is used to indicate that clicking on this object will cause form_do to return to your program, with the index of this object as its result. If the Exit bit is not set the user can continue to edit the Form.

This bit can be used for any type of object, but only with Buttons is the size of the border increased to indicate to the user that this is an Exit Button. When you create a new Button using the Object menu this bit will be set.

The Selected bit should be set whenever the Exit bit is set.

Editable

The Editable bit should only be set for the FText and FBoxText objects; this indicates that the user may edit the text in this field. If you set this bit for other types of objects the AES will mis-behave often causing the system to bomb. The fields in the TEDINFO structure used by the AES for FText and FBoxText objects must conform to strict rules as described under FBoxText.

There is no need to set other flags in conjunction with the Editable flag.

Do not use an Editable text field as the *last* object in a Form; all the current versions of the operating system will crash if you press cursor down when editing this field.

Radio Button

The Radio Button bit is used to indicate that an object is one of a set of radio buttons. The objects need not be of type Button.

Every sibling of the object should have the Radio Button bit set; to ensure this you can use an IBox to surround just the objects that you wish to be Radio Buttons. Radio Button objects should have the Selectable flag set.

For an example of programming with Radio Buttons see the WTEST program.

Touch Exit

The Touch Exit bit is used to tell the AES to exit form_do when the user moves the mouse pointer over an item and clicks on it. The exit occurs when the mouse button is pressed down (rather than released as in the case of the Exit flag). Touch Exit also differs from Exit in that the button need not be Selectable. When using form_do, if the user double-clicks on a Touch Exit object then the top bit of the return value will be set. Even if your program is not interested in double-clicks, you still need to mask off the top bit.

This flag may be used with any kind of object.

Hide

The Hide bit is used to hide an object and all its children from the AES. This means that the object is not displayed by objc_draw and will not be found by objc_find. This is useful when you wish to remove part of a tree temporarily, without re-organising that tree. For example, WERCS itself uses this facility when drawing the Extras dialog box to ensure that only appropriate types of objects are shown.

If you have hidden an object using WERCS you cannot use the Hide command from the Flags menu to unhide it again because you cannot select it; instead select its parent and then use the UnHide Children command from the same menu instead.

Flag States

The flag states for objects are as follows.

Selected

If the Selected flag is set, it indicates that the object will be displayed highlighted. This bit is changed from 0 to 1 or from 1 to 0 if the object is Selectable when the user clicks on the appropriate object. Any type of object may have this bit set.

Crossed

The Crossed bit causes the AES to draw a white diagonal cross through the object. If the object is Selected then the cross is displayed as black. This flag can be used on all objects except IBoxes.

Checked

If the checked flag is set the AES will draw the object with a black tick mark, ✓, inside it with the tick in the top left corner. When the object is Selected the tick is shown in Black. The Checked flag may be used for any type of object including IBoxes.

Disabled

If the Disabled flag is set for an object then it is shown greyed, that is, with less intense colour than normal. In addition, Disabled objects may not be Selected when using form_do or as part of a Menu even if they have the Selectable bit set. Note, though, that Disabled Editable fields may be edited!

Outlined

If the Outlined bit is set then the object is drawn with a black box outside it. Note that this does not form part of the object as far as objc_find, for example, is concerned. This bit may be used with all types of objects.

Shadowed

If the Shadowed bit is set for an object a shadow is drawn outside the object in the object's border colour; this includes Buttons. The Shadowed bit has no effect on objects without a border.

Selecting both Outlined and Shadowed attributes produces a messy display of the object and should be avoided.

Object, Flags and States Summary

The following table shows which attributes change the appearance on screen for each type of object. Text Attr refers to the alignment and size of text:

	Fill Pattern	Fill Colour	Xparent /Opaque	Border Colour	Border Size	Text Colour	Text Attr
Box	✓	✓		✓	✓		
BoxChar	✓	✓		✓	✓	✓	
BoxText	✓	✓	✓	✓	✓	✓	✓
Button							
FBoxText	✓	✓	✓	✓	✓	✓	✓
FText			✓		✓	✓	✓
IBox				✓	✓		
Icon		✓				✓	
Image						✓	
ProgDef							
String							
Text			✓		✓	✓	✓
Title							

The following table shows the effect of particular flag/state sets for a number of the Flags. Remember that Selectable, Radio Button, TouchExit, Selected and Outlined may be used for all types of objects except Titles:

	Default	Exit	Editable	Crossed	Disabled	Shadowed
Box	☞	☞	✸	✓	☞	✓
BoxChar	☞	☞	✸	✓	✓	✓
BoxText	☞	☞	✸	✓	✓	✓
Button	✓	✓	✸	✓	✓	✓
FBoxText	☞	☞	☞	✓	✓	✓
FText	☞	☞	☞	✓	✓	✖
IBox	☞	☞	✸	✖	☞	✓
Icon	☞	☞	✸	✓	✓	✖
Image	☞	☞	✸	✓	✓	✖
ProgDef	☞	☞	✸	✓	✓	✖
String	☞	☞	✸	✓	✓	✖
Text	☞	☞	✸	✓	✓	✖
Title	☞	☞	✸	✓	✓	✖

Key:

- ✓ Changes appearance of object and the behaviour of the AES.
- ☞ Changes the behaviour of the AES but not the appearance.
- ✖ Has no effect.
- ✸ Causes the machine to crash with bombs.

Programming with Resources

This section details the various data structures and object types, together with common AES programming algorithms. This section uses the standard names and typedefs for the data structures and their components which are supplied in the header file aes.h.

Tree Structure

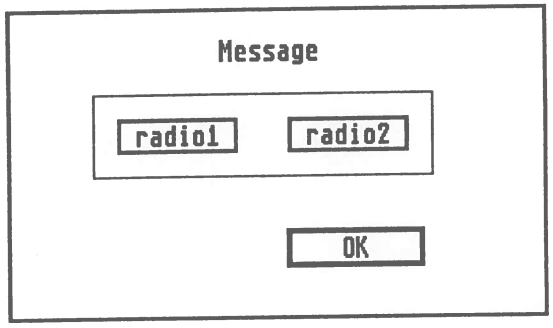
OBJECT Structure

A tree is stored in memory as an array of objects. Each object has pointers to allow the AES to tree-walk as required. The structure is as follows:

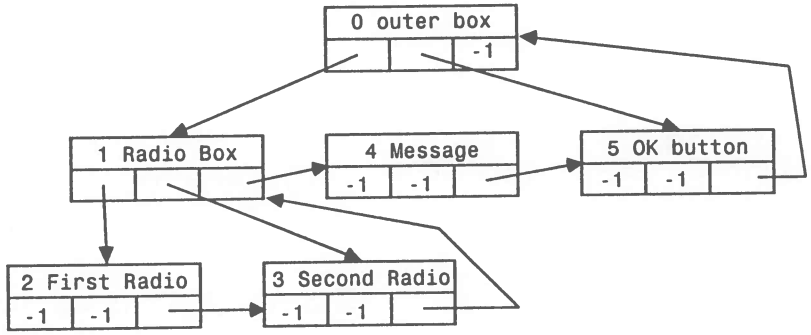
```
typedef struct object
{
  short ob_next;              index of object's next sibling
  short ob_head;              index of first child or -1 if
                              none
  short ob_tail;              index of last child or -1 if
                              none
  unsigned short ob_type;     object type (high byte is
                              ignored by the AES and used for
                              extended object numbers)
  unsigned short ob_flags;
  unsigned short ob_state;
  void *ob_spec;              depends on object type
  short ob_x;                 X co-ordinate of object
                              relative to parent (in pixels)
  short ob_y;                 Y co-ordinate of object
                              relative to parent (in pixels)
  short ob_width;             width of the object in pixels
  short ob_height;            height of the object in pixels
} OBJECT;
```

All the fields are present for all objects although the ob_spec field depends on the object type and is usually a pointer to another structure as described below.

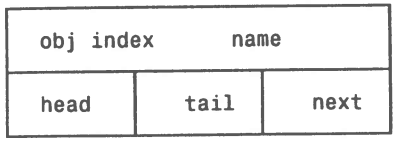
When it is loaded into memory an object tree is like an array of records. The first object (with index 0) is called the *root object*. It is normally the outer Box of a dialog box. Each object in the tree has three fields called ob_head, ob_tail and ob_next. These hold integer values that dictate to the AES the structure of the tree. Fortunately you do not normally need to access these directly, WERCS does it for you. As an example, say we have a dialog box like this:



The tree structure this represents can be shown as:



where each box represents:



When this is stored in memory the index and first three fields of the various objects will be:

Index	ob_head	ob_tail	ob_next	name
0	1	5	-1	outer box
1	2	3	4	Radio Box
2	-1	-1	3	First Radio
3	-1	-1	1	Second Radio
4	-1	-1	5	Message
5	-1	-1	0	OK button

Object number 0 is called the *root* of the tree. Its *children* are Message, Radio Box and OK button. Radio box's *parent* is outer box; its children are First Radio and Second Radio; its *siblings* are Message and OK button. First Radio and Second Radio are *childless* and they are *grand children* of the root object, outer box.

Normally what is important with object trees is the tree structure not the order that the items are in memory. For example, don't assume that the first child immediately follows its parent.

We can now give concise definitions of the ob_head, ob_tail and ob_next fields:

ob_head — points to the first child (or is -1 if childless)

ob_tail — points to the last child (or is -1 if childless)

ob_next — points to the next sibling or, if there are no more siblings, to the object's parent.

Values for the ob_type field, together with the interpretation of the ob_spec field will now be described, followed by the definition of the data structures to which they refer.

Box G_BOX 20

The ob_spec field contains the colour word (low word) and border thickness (high word).

Text G_TEXT 21

The ob_spec field contains a pointer to a TEDINFO structure; the te_ptext pointer within the structure points to the actual displayed text.

BoxText G_BOXTEXT 22

The ob_spec field contains a pointer to a TEDINFO structure; the te_ptext pointer within the structure points to the actual displayed text.

Image G_IMAGE 23

The ob_spec field contains a pointer to a BITBLT structure.

ProgDef G_PROGDEF 24

The ob_spec field contains a pointer to an APPLBLK structure.

IBox G_IBOX 25

The ob_spec field contains a colour word (low word, only border colour attribute used) and border thickness (high word).

Button G_BUTTON 26

The ob_spec field contains a pointer to the displayed string.

BoxChar G_BOXCHAR 27

The ob_spec field is used for the following fields:

bits 24-31 ASCII value of displayed character
bits 16-23 border size
bits 0-15 colour word

String G_STRING 28

The ob_spec field contains a pointer to the displayed string.

FText G_FTEXT 29

The ob_spec field contains a pointer to a TEDINFO structure. The text pointed to by te_ptext is merged with the template pointed to by te_ptmplt before display. The fill attributes in te_color are ignored.

FBoxText G_FBOXTEXT 30

The ob_spec field contains a pointer to a TEDINFO structure. The text pointed to by te_ptext is merged with the template pointed to by te_ptmplt before display.

Icon G_ICON 31

The ob_spec field contains a pointer to an ICONBLK structure.

Title G_TITLE 32

The ob_spec field contains a pointer to the displayed string.

Object Flags

The various flags in the ob_flags field have the following values as bits and as a hexadecimal mask:

Name on Menu	Standard Name	Bit	Mask
Selectable	SELECTABLE	0	0x1
Default	DEFAULT	1	0x2
Exit	EXIT	2	0x4
Editable	EDITABLE	3	0x8
Radio Button	RBUTTON	4	0x10
	LASTOB	5	0x20
Touch Exit	TOUCHEXIT	6	0x40
Hide	HIDETREE	7	0x80
	INDIRECT	8	0x100

The LASTOB bit is used by the AES to find the last object in an object tree; it is set for the last object and the last object alone. This bit is handled by WERCS for you but you may find it useful to access it if you write routines to manipulate trees in memory.

If the INDIRECT bit is set, the ob_spec field is treated as a *pointer* to the ob_spec field rather than the value itself. WERCS does not allow you to set this bit; if you need it then your program should set it and re-initialise the ob_spec field as required.

Object States

The following table gives the values as bits and masks of the ob_state field.

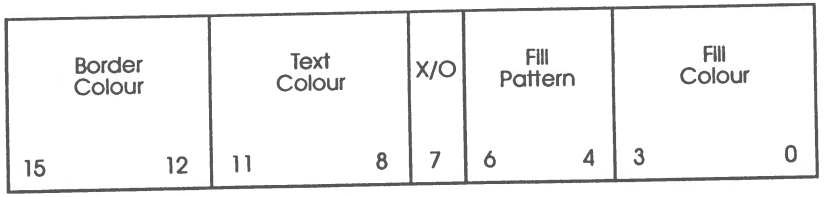
Name on Menu	Standard Name	Bit	Mask
Selected	SELECTED	0	0x1
Crossed	CROSSED	1	0x2
Checked	CHECKED	2	0x4
Disabled	DISABLED	3	0x8
Outlined	OUTLINED	4	0x10
Shadowed	SHADOWED	5	0x20

Border Thickness

The low byte of the high word in some ob_spec fields stores the border thickness in pixels. A value of 0 means no border, positive values give a border inside the object, negative values force it outside the object.

Colour Word

The colour word used in some ob_spec fields consists of the following components:



In the above diagram the numbers indicate the bits, so that the Border Colour is in bits 15-12, the four most significant bits of the first byte.

X/O is the Transparent/Opaque bit; Opaque is indicated by the bit being set.

Fill Pattern is as on the Fill menu with 0 indicating hollow- and 7 solid- fill.

The Border, Text and Fill Colours are as on the appropriate menus. The standard names for the colours are:

Colour	Value	Colour	Value
WHITE	0	LWHITE	8
BLACK	1	LBLACK	9
RED	2	LRED	10
GREEN	3	LGREEN	11
BLUE	4	LBLUE	12
CYAN	5	LCYAN	13
YELLOW	6	LYELLOW	14
MAGENTA	7	LMAGENTA	15

The L in the above names indicates *light*. If you *must* encode a colour word into your program the best base to use is hexadecimal.

TEDINFO Structure

This structure is used by the object types BoxText, FBoxText, FText and Text:

```
typedef struct text_edinfo
{
  char *te_ptext;       pointer to actual text
  char *te_ptmplt;      pointer to template; editable
                        portion denoted by underscores
  char *te_pvalid;      pointer to string containing
                        validation characters
  short te_font;        font used: 3=system font, 5=small
                        font
  short te_junk1;       reserved for future use
  short te_just;        text justification required:
                        0=left, 1=right, 2=centre
  short te_color;       object colour and pattern of box-
                        type objects (see previously for
                        word format)
  short te_junk2;       reserved for future use
  short te_thickness;   border thickness
  short te_txtlen;      length of te_ptext string
                        (including null)
  short te_tmplen;      length of te_ptmplt string
                        (including null)
} TEDINFO;
```

ICONBLK Structure

This is used by the Icon object type only:

```
typedef struct icon_block
{
  short *ib_pmask;      pointer to icon mask
  short *ib_pdata;      pointer to icon data
  char *ib_ptext;       pointer to the text displayed with
                        the icon
  short ib_char;        low byte is the displayed character,
                        high byte defines colour used - top
                        nibble is foreground colour, bottom
                        nibble is background
  short ib_xchar;       X co-ordinate of ib_char relative to
                        ib_xicon
  short ib_ychar;       Y co-ordinate of ib_char relative to
                        ib_yicon
  short ib_xicon;       X co-ordinate of icon relative to the
                        ob_x of the object
  short ib_yicon;       Y co-ordinate of icon relative to the
                        ob_y of the object
  short ib_wicon;       width of the icon image in pixels
                        (must be a multiple of 16)
  short ib_hicon;       height of icon image in pixels
  short ib_xtext;       X co-ordinate of icon's text relative
                        to the ob_x of the object
  short ib_ytext;       Y co-ordinate of icon's text relative
                        to the ob_y of the object
  short ib_wtext;       width of rectangle to display icon's
                        text in (centred)
  short ib_htext;       height of icon's text
} ICONBLK;
```

The bit images for the mask and data are stored as arrays of words.

BITBLK Structure

This is used by the Image object type and Free Images only:

```
typedef struct bit_block
{
  short *bi_pdata;   pointer to bit image
```

```
    short bi_wb;        width of image data in bytes (must be
                        even)
    short bi_hl;        height of image in pixels
    short bi_x;         source X co-ordinate
    short bi_y;         source Y co-ordinate
    short bi_color;     colour word (see previously)
} BITBLK;
```

bl_x and bl_y are used as offsets into the bit image given by bi_pdata; any bits before this will be ignored.

APPLBLK Structure

This is used by ProgDefs:

```
typedef struct appl_blk
{
  int (*ab_code)(PARMBLK *);     pointer to code to draw the
                                 object
  long ab_parm; passed as a parameter to the low-level
                                 drawing routine
} APPLBLK;
```

PARMBLK Structure

This is passed to ProgDef drawing routines:

```
typedef struct parm_blk
{
  OBJECT *pb_tree;        pointer to start of object tree
  short pb_obj;           the object index
  short pb_prevstate;     the old state of the object to be
                          changed
  short pb_currstate;     the new (changed) state of the
                          object
  short pb_x;             the pixel X screen co-ordinate of
                          the object
  short pb_y;             the pixel Y screen co-ordinate of
                          the object
  short pb_w;             the pixel width the object
  short pb_h;             the pixel height of the object
  short pb_xc;            the pixel X co-ordinate of the
                          current clip rectangle
  short pb_yc;            the pixel Y screen co-ordinate of
                          the current clip rectangle
  short pb_wc;            the pixel width of the current clip
                          rectangle
  short pb_hc;            the pixel height of the current
                          clip rectangle
  long pb_parm;           copied from the ab_parm value in
                          the APPLBLK
} PARMBLK;
```

If pb_prevstate and pb_currstate are the same then the AES is drawing the object, not changing it.

Hints & Tips on Resources

Using ProgDefs

If a loaded resource file contains any ProgDef objects, their ob_spec; field will not be initialised on loading - this is up to the programmer. An APPLBLK structure needs to be allocated and initialised, then a pointer to it planted in the relevant ob_spec field.

The drawing routine (in the ab_code field) will then be called whenever that object needs drawing or changing (remember that if you have a ProgDef in a menu this may occur at *any* time). The routine called should normally be declared as both __stdargs and __saveds, taking a single parameter pointing to a PARMBLK. Hence a typical declaration would be:

```
int __stdargs __saveds my_progdef(PARMBLK *pb);
```

When your custom drawing routine has finished, the value it returns is the ob_state value which you wish the AES to render over your object, i.e. returning a value of 0 applies no extra effects, whereas returning CROSSED (for instance) would draw a cross over the object. Note that any number of ob_state values may be ORed together to produce the desired effect..

When designing ProgDefs it is often easiest to base them on existing objects which can be manipulated in WERCS, e.g. in the example program we implement a rounded button based on the normal square button, hence the text may be manipulated from within WERCS.

When the AES calls your drawing code you are still in the AES's 'context', i.e. you are using its stack, hence recursive routines or large local arrays may cause it to overflow. Note that this also means that the routine which is called *must* be compiled with stack checks *off* (the -v option). Also note that the AES is *not* re-entrant hence you may not make any calls to it (although you can, and should, call the VDI).

If you draw using the AES's handle (as in our example) then you should ensure that you maintain any of the VDI attributes, alternatively you may use your own virtual workstation which will avoid these problems.

Creating New Desktops

It is possible to replace the standard GEM background pattern (the area of the screen not used by the menu bar) using a special tree. This allows different colours and fill patterns to be used, as well as allowing icons to appear on the desktop.

A Form should be created in WERCS with the root object being a borderless Box with a suitable fill pattern and colour. If any icons are required these should be added to this Form. The size of the Form is not relevant. To tell GEM to use this Form, the size and position (ob_x, ob_y, ob_width and ob_height) fields in the root object should be set to the usable screen size, found using the AES wind_get(DESK, WF_WORKXYWH, ... call. The form can then be installed using a wind_set(WF_NEWDESK, ... call with an object parameter of zero. Before your program terminates, the desktop must be de-installed by passing a NULL value to the same call.

Note that installing a desktop does not cause it to be drawn and you should normally call form_do(FMD_FINISH, ... to force a redraw of the area.

Common Mistakes and how to avoid them

The following is a list of common mistakes made when programming with GEM and resources in general. The reasons given here are brief as there is insufficient space to expand upon them; they act as pointers for where to look in other documentation.

Problem: My program was mainly working but now it crashes during its initialisation.

Reason: Your resource file is out of step with your program and what was the Menu that you were displaying is now a Form; as a result the GEM menu_bar call bombs. Re-compile all the parts of your program that rely on the header file.

Problem: My dialog box doesn't disappear after you click on OK.

Reason: The dialog box is on top of one of your windows and you are not replying to WM_REDRAW events. If you don't open a window, the Desktop will re-draw the desktop tree for you automatically .

Problem: My program crashes when it should be displaying a Dialog Box.

Reason 1: If you have no editable fields and are passing -1 as the starting object the machine may crash, despite what some documentation says. Use 0 instead.

Reason 2: If you do have editable text fields make sure that they conform to the rules under FBoxText regarding editable text.

Problem: A GEM program crashes unexpectedly. After rebooting, the same program works correctly under the same conditions.

Reason: A program has modified GEM's data structures unintentionally. There are many possible ways of doing this; one to look out for is not doing a v_clsvwk after a v_opnvwk; that is leaving a Virtual Workstation open.

Problem: The mouse disappears or leaves extra pixels on the screen ('mouse droppings').

Reason: Your graf_mouse calls are mis-balanced in some way. For each hide (M_ON) call you *must* have a show (M_OFF) call.

Problem: There are mouse droppings where a menu has been pulled down.

Reason: You are not using wind_update (BEG_UPDATE, ... and graf_mouse (or the VDI v_hide_c) before writing to the screen.

Problem: When using some desk accessories the screen display is messed up.

Reason: Make sure that you are taking note of WM_REDRAW events and only updating the areas given by the wind_get (WF_FIRSTXYWH, ... and wind_get (WF_NEXTXYWH, ... calls. To test this, move a desk accessory about the screen; the Control Panel and the Saved! desk accessory can both be used.

Problem: Some desk accessories 'lose' their mouse when invoked from my program.

Reason: Make sure you don't remove the mouse until *after* you have done a wind_update (BEG_UPDATE, ... call and make sure that it is visible before calling wind_update for END_UPDATE.

Problem: The program works fine in medium and high resolution, but crashes when accessing a menu on 'old' ROM machine.

Reason: Your menu is taking up more than one quarter of the screen. When running in Low Resolution, a menu may not contain more than 16000 pixels. If you are using large menus, you may wish to consider using a special menu for low resolution, as WERCS does.