301 System error <val> on write

A system error occurred whilst attempting to write to the disk. This will normally indicate that the disk is full. The value of the error is given by <val>.

400 *** Break: Clink terminating

This message is printed when the operation of the linker is interrupted by the user pressing Ctrl-C. Note that the keyboard is only checked whilst screen input or output is occurring.

425 Cannot find library <file>

The file named in a LIB statement could not be located by the linker. This is probably due to a full pathname not being given for the file.

426 Cannot find object <file>

The file named in a FROM or ROOT statement could not be located by the linker. This is probably due to a full pathname not being given for the file.

443 '<file>' is an invalid file name

The filename specified in a FROM, LIB or ROOT statement is invalid. Typically this will be because the name is null.

444 hunk_symbol has bad <val> symbol <file>

A hunk_symbol hunk type was encountered by the linker which did not have the external type set to zero, but instead to val. If this error occurs it indicates that the named input file was damaged in some manner.

445 Invalid hunk_symbol <name>

A hunk_symbol hunk type was encountered by the linker during parsing of the external definitions. The named symbol was attached to this hunk.

446 Invalid symbol type <val> for <file>

Whilst parsing external declarations an unknown symbol type <val> was encountered in the named file.

448 <file> is not a valid object file

The named file did not match the specifications for an object module.

449 No hunk_end seen for <file>

On reaching the end of a hunk within the named file an end marker did not appear.

450 Object file <file> is an extended library

An attempt has been made to use a library as the operand of a FROM or ROOT statement. Libraries may only be searched, not included.

501 Invalid Reloc 8 or 16 reference

An attempt has been made to generate a branch between two differently named sections. Branches may only occur within a common section. This error will normally indicate an attempt to execute the data section!

502 <name> symbol - Distance for Reloc16 > 32768

The target of a 16 bit branch is more than 32K away from the reference. In general you should not see this message due to ALV generation.

503 <name> symbol - Distance for Reloc8 > 128

The target of an 8 bit branch is more than 128 bytes away from the reference. Note that the compiler does not generate such external branches and that the assembler does not allow their generation.

504 <name> symbol - Distance for Data Reloc16 > 32768

A 16 bit base-relative data section access is attempting to reach more than 32K. This error will normally indicate you are very close to the 64K limit on near data, and a a module has had its data section fall off the end of the merged data section (biassed by 32K). The solution is to reorder your modules putting the ones with large data sections alternatively you may have to move some of your near data to far.

505 <name> symbol - Distance for Data Reloc8 > 128

An 8 bit base-relative data section access is attempting to reach more than 128 bytes. This error will normally indicate incorrect code generation from the compiler.

506 Can't locate resolved symbol <name>

During the second pass the linker could not locate the named symbol in its table. This will either indicate an internal linker failure or a damaged library file.

507 Unknown Symbol type <val>, for symbol <name>

During the second pass the linker could not match the type of the named symbol in its table. This will indicate an internal linker failure.

508 Symbol type <val> unimplemented

Whilst parsing external declarations an unknown symbol type <val> was encountered in the named file. Note that the equivalent error (446) is reported during pass 1.

509 Unknown hunk type <val> in Pass2

The named file did not match the specifications for an object module. Note that this message is identical to the pass 1 error 448.

510 <name> symbol - Reference to unmerged data item

A module has attempted to access an unmerged (far) data item using a near access. This will usually occur if one module of a program is compiled using the -b1 option whilst another module uses -b0. The error message also suggest the action: try -b0 option on LC.

515 An ALV was generated pointing to data <name> symbol

An ALV was generated in the data section of the program. This will only occur if code generation has been performed in a data section, and as such this error will normally indicate an internal compiler failure.

600 Invalid command '<cmd>'

The named command was not recognised by the linker. The commands which are recognised are discussed in the section **CLink, The Linker.**

601 <cmd> option specified more than once

An attempt has been made to specify a command, which may only appear once, more than once, e.g. attempting to specify two TO files.

602 Unable to open output file '<file>'

The named output file could not be opened. This may be because the disk or directory is full.

603 <val> is not a valid number

The value <val> which appeared as a numeric argument could not be parsed as such.

604 with file is not readable

An error occurred whilst reading the WITH file.

605 Cannot open with file '<file>'

The named WITH file could not be opened.

607 No FROM files specified

No FROM or ROOT files were specified so the linker cannot start linking.

608 Premature EOF encountered

End-of-file occurred unexpectedly. This will normally indicate serious file system structure problems.

609 Error seeking in file <file>

An error occurred whilst attempting to seek about the named file. This will normally indicate serious file system structure problems.

611 Reloc found with odd address for symbol <name>, file <file>

A 16 or 32 bit relocation was attempted on a non word-aligned boundary. This is always illegal on the 68000.

ERROR: Invalid decimal constant '<val>'.

The value <val> which was entered in response to an undefined symbol was an invalid decimal constant.

ERROR: Invalid hex constant '<val>'.

The value <val> which was entered in response to an undefined symbol was an invalid hexadecimal constant.

ERROR: Multiply defined symbol '<name>'.

A symbol has been redefined. The file in which it first appears is named, as is the file in which the attempted re-definition occurs.

ERROR: Symbol '<name>' is not defined.

The named symbol which was entered in response to an undefined symbol was also undefined.

Hunk #n not written

The numbered hunk n was not written to disk. This will indicate an internal linker failure.

Unknown internal error

An internal error occurred whose error number was not recognised. This indicates a serious internal linker failure.

Batcher
The Command Shell

Batcher is a command line processor modelled on the COMMAND.COM processor of MSDOS. It can be used instead of the GEM icon interface if you like, or it can be used to run batch files. Unlike some other command line interpreters on the ST, you can still run GEM programs (such as EdC) from Batcher.

To run a program under Batcher, give the file name for the program. If you do not give an extension and a file without an extension cannot be found, Batcher will try to load a file with the extension .PRG, then .TOS and finally .TTP.

There are a number of commands built-in to Batcher i.e. no other program has to be loaded in order that they can be executed. These are described in the following sections.

Batcher can also be used to set up environment variable values used to configure the compiler, and to tell Batcher where to look for programs.

First we will describe the commands that are built in to Batcher and then we will discuss its line editing and batch file facilities.

AVAIL — Available memory

AVAIL displays the size of the largest free GEMDOS block of memory.

CD — Change Directory

The CD command changes the current directory. Directories are also known as *folders*.

This command takes one parameter; the name of the directory to go to. The syntax of the directory name follows the standard syntax used for path names. If the first character is a backslash (\) then the path is relative to the root directory. Otherwise, it is relative to the current directory. In addition, .. refers to the directory one level towards the root directory from the current directory.

Note that, following the operating system convention, there is a separate 'current' directory for each drive. This means that, to move to a particular directory, you should first use the Change Disk command described below and then use the CD command. For example to move to C:\LC you would use:

```
C:
CD \LC
```

This scheme has a useful advantage. Say you have two versions of a program in E:\MINE\OLD and D:\MINE\NEW. Then after using:

```
CD E:\MINE\OLD
```

and

```
CD D:\MINE\NEW
```

you can use E: and D: to refer to these two directories.

B: — Change Disk

B: (or b:) makes disk B the current disk. Any drive letter may be used so that whenever you reference a file without giving an explicit disk designator, the current disk will be used.

CLS — Clear screen

This command clears the screen and enables line wrap. If the COLOUR command has been used then the screen colours will be set as per the last colour command. CLS is useful of your program has left you with black text on a black background!

COLOUR — Set screen colours

This command takes two numeric parameters which set the background and foreground screen colours, using TOS rather than GEM colours, as shown in the table below

Number	High	Medium
0	White	White
1	Black	Red
2		Green
3		Black

e.g. If you are using mono-chrome:

```
COLOUR 0 1
```

would use black text on a white background. If you are using medium resolution:

```
COLOUR 3 1
```

would give red characters on a black background.

COPY — Copy Files

Copy files. You can copy single files or groups of files. You can use wildcards in the source and destination names. For example, to copy all the .C files from disk A to disk D enter the command:

```
COPY a:*.C D:
```

To make a backup code of all the .C files whose name starts with the letter F, you could use:

```
copy f*.c *.BAK
```

Note that the f is omitted in the second file specification.

You can ask to be prompted for the files to be copied by specifying the /a (for ask) flag. e.g.

```
COPY d:\myfiles\*.o a:\ /a
```

will prompt you with the names of all the .o files in the d:\myfiles directory.

You may reply:

Y	copy this file
N	ignore this file
Q or Ctrl-C	quit and don't copy any files
A	copy this and the remainder of the files

You can also modify the order in which the files are listed by using the /S (for size order) or /D (for date order) flags, optionally followed by a - to reverse the order; so

```
copy \fred\*.c a: /ad
```

would prompt you for the .c files in the fred directory on the root of the current drive, in date order.

Notice that you may enter commands and arguments in either upper or lower case.

Copy will use as much free memory as it can and read in as many files as possible. This means that copying files is often considerably faster than with the Desktop, especially on a single floppy system.

COPYWARN — Enable file overwrite warnings

The COPYWARN command lets you enable warnings when over-writing an existing file using the COPY command. Use

```
COPYWARN ON
```

to enable this. If you are prompted, press Y to over-write this file, N to leave this file and A to copy this and all subsequent files without asking.

DEL — Delete Files

DEL deletes files. You can use a wild card specification and Batcher will prompt you before it deletes each file to ensure that you really want to delete it. Type Y to delete this file; N not to delete it; Q to quit the delete command and A to delete the remaining files.

To avoid the prompting entirely, use the N flag:

```
DEL *.PRG/N
```

You can also modify the order in which the files are listed by using the /S (for size order) or /D (for date order) flags, optionally followed by a - to reverse the order.

DC — Disk Change

The ST's operating system can fail to notice that a floppy disk has been changed with the potentially disastrous consequence of corrupting the floppy. This is especially a problem when using disks with identical serial numbers under TOS 1.4 and 1.6.

The DC command can be used to ensure that the operating system notices that you have changed a floppy disk. It may be followed by a drive letter, but by default it will apply to drive A. Thus

```
DC
```

will ensure that the operating system notices that the disk in drive A has been changed and

```
DC B:
```

will inform the operating system that the disk in drive B has changed. If you find that use this command frequently, then you should consider using the DISKCHANGE command.

DIR — Directory List

This commands lists the files in a directory. You may follow DIR by a directory name or a drive name or a wild-card file name for which files to display. For example:

```
DIR headers
DIR B:
DIR d:\lc\*.c
```

If you specify the W flag, only the file names are displayed. Otherwise, the names, sizes and dates for the files are displayed.

```
DIR d:\sources\*.c/W
```

You can also modify the order in which the files are listed by using the /S (for size order) or /D (for date order) flags, optionally followed by a - to reverse the order:

```
DIR *.h/D
```

lists all the .H files in the current directory, in date order.

DISKCHANGE — Set auto-diskchange mode

Using

```
DISKCHANGE ON
```

causes Batcher to ensure that the operating system has noticed that a disk has changed whenever you use a built-in command that refers to a floppy disk drive. As such it takes a second or two before the command is executed. We thus recommend that it is not used on floppy only systems unless you are changing disks a great deal. Its effect can be disabled by using:

```
DISKCHANGE OFF
```

which is the default. You can ensure that the operating system notices a particular disk change by using the DC command, see above.

ECHO — Echo commands

Echo commands as they are processed. When echoing is on, each command in a batch file will be displayed on the screen before it is executed. You can use

```
ECHO OFF
```

to prevent the commands in batch files being echoed as they are performed and ECHO ON to turn this back on.

If the parameter to ECHO is not ON or OFF then the text will be echoed to the screen. For example:

```
ECHO This is a message
```

will display 'This is a message'.

ERA — Erase Files

This command is exactly the same as DEL and is supplied for the benefit of CP/M users.

EXIT — Exit Batcher

This command exits Batcher. You do not need to use this in .BAT files since Batcher will exit when it comes to the end of the batch file.

FORMAT — Format floppy disk

This will format standard ST double-sided disks without any interleave. It should be followed by the drive to format e.g.

```
FORMAT B:
```

will format the disk in drive B. A second (optional) parameter may be supplied giving the volume label which is to be given to the disk, e.g.

```
FORMAT B: VOLID
```

Formats a disk in drive B with a volume label of VOLID.

FREE — Free disk space

Returns the free space (in bytes) on a disk. For example,

```
FREE
```

gives the free space on the current disk while:

```
FREE A:
```

gives the free space on drive A.

MKDIR — Make Directory

This command creates a new directory (or folder) with the given name. You may use a full path specification if you wish, although MKDIR will not attempt to create more than one directory at once.

```
MKDIR SOURCES
```

will create a directory called SOURCES on the current disk.

MOUSE — Control Mouse Visibility

Turns the mouse on (use MOUSE ON) or off (MOUSE OFF). This is useful if you run a program which leaves the mouse in a funny state. Whenever you run a .PRG program, CLI enables the mouse. It disables the mouse on return. If a program disables the mouse and fails to re-enable it before it exits, subsequent programs will not show the mouse cursor.

Conversely, entering MOUSE OFF in Batcher will prevent the mouse cursor appearing when you run a program, which can be useful when running non-GEM applications that have extensions of .PRG.

When Batcher runs programs with extensions of .TOS and .TTP the mouse is not enabled.

To set the mouse back to the default value, issue MOUSE ON commands until the mouse appears and then issue MOUSE OFF commands until it vanishes again.

PAUSE — Pause for keypress

PAUSE will wait for a single key to be pressed. This can be used in batch files so that the user can read the previous output or change disks.

REM — Remark

Used to place a comment in a batch file - the line is ignored.

REN — Rename

Rename files. Wild cards can be used:

```
REN *.O *.OO
```

will rename all the files in the current directory with extension .O to have extension .OO.

RMDIR — Remove Directory

This command deletes a directory (or folder) with the given name. The directory must be empty before you can remove it. You may use a full path specification if you wish.

```
RMDIR A:\SOURCES
```

will delete a directory called SOURCES on the current disk if it does not contain any files.

SCREENSAVE — Set screensave mode

When turned on (SCREENSAVE ON) the screen is not cleared before a .PRG file is run. This is useful when running .PRG files that are not really GEM applications, so that you can see the output from previous commands.

SET — Set Environment Variable

Sets environment variables. If you just type SET, the current environment variable values are listed.

```
SET INCLUDE=d:\headers
SET PATH=d:\lc;c:\bin
```

You can remove a variable by setting it equal to no value. For example,

```
SET QUAD=
```

removes the QUAD environment variable.The environment variables used by Lattice C are described under **LC, The Compiler**.

SMALL — Set font size

This command is only really useful in monochrome.

```
SMALL ON
```

will cause TOS output, like that from Batcher, to appear in the 8x8 system font. On a high resolution screen, this gives 50 lines on the screen.

```
SMALL OFF
```

selects the 8x16 font giving the usual 25 lines in monochrome, but only 12 in medium resolution.

TYPE — Type File

TYPE displays a file on the screen.

```
TYPE HELLO.C
```

VIRTUALDISK — Perform virtual disking

This command is used by the installation program so that Batcher can perform 'virtual disking' itself rather than using the operating system routines, which do not work reliably. However, as all Batcher commands do not support this facility, we do not recommend that it is used interactively.

WHICH — Which file would run

This command returns the path of the file that will be run if you use a given filename. For example,

```
WHICH WERCS
```

will tell you from where WERCS will be loaded if you type WERCS to Batcher. This can be very useful if it appears that you are running the wrong program, particularly if it is called TEST !

Line Editing

When using Batcher you can also use the cursor keys. The ← and → keys will move the cursor within the current line, whilst you can us the ↑ key to display the previous command that you entered to Batcher. This is very useful if you have made a simple typing mistake. The area of memory where these commands are stored is known as the *history buffer*. You may press ↑ more than once; this will display the other lines that you entered previously. If you go too far back in the history buffer, then press ↓ and then commands will be displayed in the order in which they were entered.

Pressing Backspace will delete the character to the left of the cursor; Delete will delete the character under the cursor. Pressing Ctrl and ← will take you to the beginning of the current line and Ctrl and → will take you to the end of the line.

You can also recall the last line that starts with a particular character sequence by typing ^ first. For example:

```
^lc [Return]
```

will display the last line that started with LC and you can then edit this line if required. Using a prefix of ! will cause the last such command to be run. For example:

```
!cl[Return]
```


will probably run the last CLInk command.

Batcher also supports some of the traditional MS-DOS line editing keys:

F1	copy one character from the previous line
F3	copy the rest of the previous line
F5	clear the current line
Delete	ignore one character from the previous line
Insert	cause the following characters to be inserted. Pressing Insert again switches this off.

The best way to learn about these features is to experiment.

Batch file facilities

Batch files are very useful for re-compiling groups of files. For example, you might have a single batch file that re-compiles (and re-assembles) every module in a large program. A batch file consists simply of the list of commands that you wish to execute and has the extension .BAT. To run a batch file, just type its name from within Batcher. If you are using the GEM Desktop you can just double-click on the batch file, if you have installed Batcher for the document type .BAT.

When Batcher is executed without a command line, it looks for a batch file called AUTOEXEC.BAT and executes the commands contained in this file. The most common use of this facility is to set up environment variables, and your preferences, but you can use it for any purpose.

Batch files may have parameters. Within the batch file these are referenced as %1, %2 up to %9 for the last parameter. To include a % character in a batch file use %%.

For example if mv.bat contained

```
del %2
ren %1 %2
```

Then:

```
mv  mine.c new.c
```

would rename mine.c to new.c deleting any old version of new.c.

If the first two non-space characters on Batcher's command line consist of /C, Batcher treats the rest of its command line as a single command. This facility is provided so that you can use Lattice C system function to run Batcher commands from within your own programs. You need to set the COMSPEC environment variable so that the system function will know where to find Batcher.

Redirection

Batcher supports command line re-direction. To cause the output from a program that would normally be sent to the screen via GEMDOS to be sent to a file, add > (greater than) and the name of the file. e.g

```
dir *.c >mydir
```

will create a list of the C files in the current directory to the file mydir. You can also append to an existing file by using >>, thus:

```
dir *.h >>mydir
```

would add the .h files in the current directory to the end of the file, mydir. You can also re-direct input to come from a file using <.

Note that all redirection operators must be at the end of the command line and that due to problems with the operating system, some programs may behave strangely if their input/output is re-directed. Some of these anomalies are described in the **Volume III - Atari Library manual**.

WERCS
The Resource Editor

WERCS is an acronym for WIMP Environment Resource Construction Set and is pronounced *Works*. It allows you to create and edit resource files for use with GEM programs.

What is a Resource File?

A resource file is a special file (normally with the extension .RSC) that contains *resources*. A resource is actually a *tree* structure in memory which is used by the GEM AES to produce such things as:

- Menus
- Dialog boxes
- Icons
- Alert boxes
- Strings

A resource file contains such things to deliberately keep them separate from your program code. In addition, the X-Y co-ordinates of every item in a tree is stored in such a way as to produce the same visual layout, regardless of the screen resolution. This means one resource file can be used for all screen modes and by many different programs.

Using resource files is good practice because it encourages modularity and aids portability thus saving you time and energy in the long run.

A certain understanding of the way a resource file works is required in order to create and use such a file.

Each resource file contains one or more *trees*. A tree may be one of five different types: *Form*, *Menu*, *Free String*, *Alert* or *Free Image*. Forms and Menus are the most common; each of these, in turn, consists of individual *objects*, where each object has a distinct type, use, purpose and appearance.

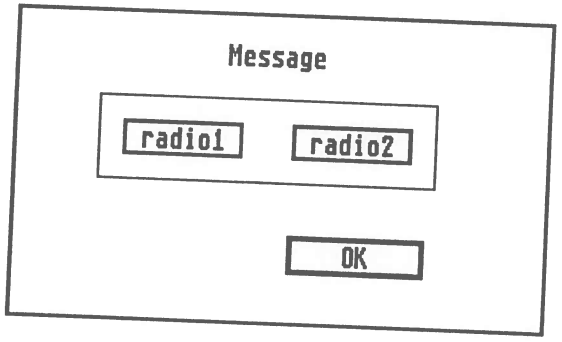
Whilst you are learning to use WERCS we recommend that you start off by using just Forms.

What is a Tree ?

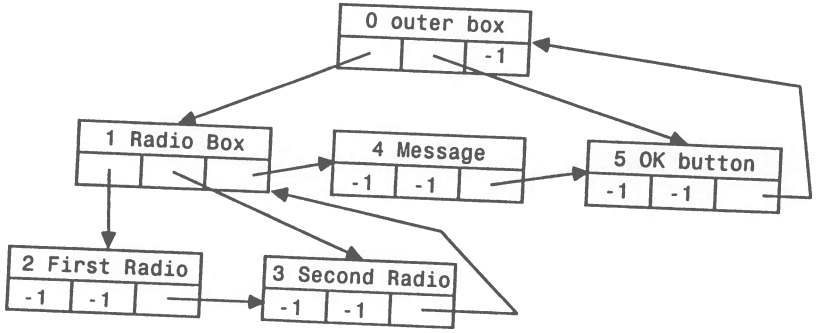
Forms (or Dialog Boxes) and Menus are GEM AES *object trees* and to understand resource files you need to understand the structure of object trees.

Many of the WERCS commands work on parts of object trees and we shall use tree terminology to describe them.

When it is loaded into memory an object tree is like an array of records, each record describing an object. 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 eleven fields. Three of these fields, the *head*, *tail* and *next* fields, 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 the components of each box are:

obj index	name	
head	tail	next

Most of the terminology used to describe object trees is similar to that used in human family trees; of course objects only have one parent and most people don't think of themselves as ultimately descended from a root!

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. The detail of how trees are stored in memory is described in **Appendix B - Resource Details**.

What is an Object?

There are thirteen types of object that you can have in object trees; most of them are some form of text or boxes or a combination of both. Different types have different memory requirements; in general the more flexibility the more bytes are used.

As well as the fields described above, associated with each object is its *position*, *size* and also some *flags* and *states*.

The position of an object is always given relative to its parent; normally you set the position and size of the object using the mouse - WERCS takes care of the calculations for you.

The flags and states are used for two purposes; first to change the appearance of an item; for example whether a box has an outline (Outlined), and also to give information to the AES; for example that clicking on a Button will cause control to be transferred back to your program (Exit). To start off with you need not be too concerned about flags and states as WERCS gives you sensible defaults.

The thirteen types of objects are as follows:

- Box — A straightforward box - can have a fill pattern and a border.
- IBox — An 'invisible' Box; only truly invisible if it has no border.
- String — A straightforward string of characters.
- Button — Like a String but with a box round it; normally used for Dialog Box buttons.
- Text — Like a String but with more formatting possibilities: colour, size and justification.
- BoxText — Like Text but with a surrounding box as well.
- BoxChar — A single character in a box. The most memory efficient way to have a single character in a filled or coloured box.
- Title — A special form of String only used in Menus.
- FText — This is like Text but can be used for editable text so that you can type in characters, numbers etc. The programming interface isn't easy but we show you how to do it in the example program.
- FBoxText — Like FText but with a box around it.
- Image — A simple bit-mapped graphic image.
- Icon — Like an Image, but with a mask so that it changes sensibly when selected and also has a character and string associated with it. Originally invented for the desktop's disk icons.
- ProgDef — A programmer-defined object with its own drawing routine. We recommend that you don't try these out until you've exhausted the possibilities of the pre-defined objects.

Header Files

In order for a program to use a resource file, the programmer must be given a method of referring to each tree and object; WERCS helps you by creating a *header file*, as well as the actual resource file. As you create a resource file you can give names to both trees and objects, so that you can refer to these names within your program. The header file contains constants which translate these names into integer values. The header file is then #included as normal into your program.

If the compiled version of the header file is out of step with the resource file, strange things will happen; this varies from slight mis-behaviour to total system crashes.

Quick Tour

Running WERCS

To run WERCS, simply double-click on the WERCS.PRG icon. WERCS also needs its .RSC and .LNG files in order to run.

There now follows a whistle-stop tour of WERCS introducing the editing facilities available. A more detailed reference section is to be found in the next section.

Low Resolution

WERCS runs in all screen modes, for maximum flexibility. When running in low-resolution, the title of each menu is reduced to the first two characters only. However, the full menu title is shown at the top of the menu box, once it has been pulled down.

Creating a New Resource File

Having loaded and executed, WERCS will display a *tree window*, labelled Untitled. A tree window displays all the trees within the file and initially this is blank since you are starting with an empty file.

When creating a new resource file it is best to select the programming language for which you require the header file before you enter any names. This can be done by selecting Language from the File menu. You can also choose whether your names will be upper-, lower- or mixed-case. Selecting the language before you start ensures you don't make any naming errors while building the file. Naturally C is the default language with this version of WERCS.

Creating a New Tree

To create a new tree you simply select a suitable type of tree from the Tree menu - this stage is known as *tree-level editing*. An icon representing this type of tree will appear in the tree window, together with a dialog box. The main point of interest for the time being is the name that you wish to call the tree - when you are happy with its name press Return and you will then be at the *object-editing* level.

Creating Objects

WERCS will now display a window showing the new tree, allowing you to add and edit new objects. To add an object, select the object type from the Object menu. The mouse will change into a representation of that object, then you should click where you require the object to be placed. To name this object, double-click on it, to move it simply drag it, or to re-size it click on its lower right-hand corner.

When you are satisfied with the objects in this tree, clicking on the Close box will return you to the main tree window. If you don't like anything you have done, the last session may be aborted by selecting Abandon Edit from the Edit menu.

When the tree window is visible, an existing tree may be edited by double-clicking on its icon. Certain attributes, such as the name, may be changed by single-clicking to produce a dialog box.

When you are happy with your resource file, ensure the correct language choice has been made then Save As the file. This will create the .RSC file containing the actual resources, a .HRD file containing your names for each item, and a header file.

Using WERCS

General

Most of the editing actions inside WERCS are obtained from menus or via the corresponding keyboard short-cut. Keyboard shortcuts are shown in each menu with a ⌥ symbol denoting the Alt key, and ^ denoting the Ctrl key. Menus that are inapplicable at a particular time are disabled. Owing to a bug in the original version (1.0) of the operating system, the titles are not disabled when using these ROMs, although naturally these commands will have no effect. There is a summary of the keyboard short-cuts at the end of this chapter.

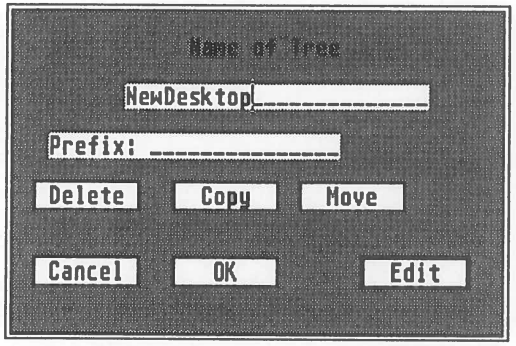
Introduction to Creating and Editing Trees

There are two main levels when running WERCS. The *Tree Level* is used for manipulating which trees are in your file and the *Object Level* for the items within those trees.

When you open a file (or use New) a window containing the trees in the file is shown, known as the *tree window*. You can add a new tree to the file by clicking on one of the items on the Tree menu: Form, Alert, Free String, Menu and Free Image. Whilst you are learning to use WERCS and if you are not familiar with GEM it is best to just use Forms. Forms account for the vast majority of trees in any case.

Note that a Form is also known as a Dialog Box; we use the terms inter-changeably. Generally we use Form in the context of editing and in the programming section we refer to Dialog Boxes.

After clicking on Form from the Tree menu you will be presented with a dialog box like this:



You can now enter the name of the tree. The defaults for these are MENU1, MENU2, FORM1, FORM2 etc.

Pressing the Return key or clicking on the Edit button will then let you edit the objects within the tree. The other buttons and fields are described in detail later.

To edit an existing tree, such as a Menu or a Form, double-click on the appropriate tree icon. You will then be taken straight to the Object Level.

Changing Objects

Once you have clicked on Edit from the Name of Tree box you are ready to add objects to the tree. To add a new item to a tree, click on the required item type from the Object menu. The mouse will change to an outline representation of the item that you are adding. Release the mouse button and move the mouse to where you would like the new item, then press the mouse button. If you decide you do not want to add this item after all, click on the Cancel item from the Edit menu.

When you have finished editing a form click on the Close box; this will return you to Tree Level mode.

Selecting objects

To change the attributes of any object, single-click on it. It will then be *selected* (highlighted) and you can use most of the menus to change its attributes, the border for example. The exceptions to this are the text items, Images and Icons; to edit these, double-click on an object.

When the object is selected the GEM *selected* bit is used to show this. This means that if you click on a box it will appear black. In particular if you click on the outer box it will all go black. If you didn't mean to click there you can either:

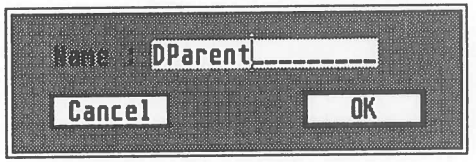
- click on the menu item Cancel from the Edit menu,
- click outside the box, or
- click on the item that you meant to select.

If you wish to edit the *parent* of the current object, single-click with the ALT key held down - the parent of the object will then be selected. If you already have an item selected and ALT-click again, its parent will be selected. This may be repeated any number of times until the whole tree is selected. To bring up the Text Box of an object's parent double-click whilst holding down ALT.

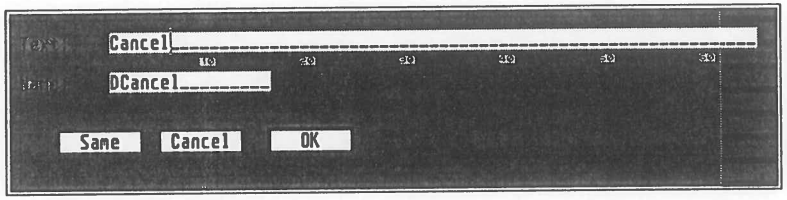
Item Names and Text

To change the Text or Name (remember: Text is the displayed message; Name is what your program will know the item as) of an object, double-click on that item - this will present a Text dialog box. This varies depending on the item.

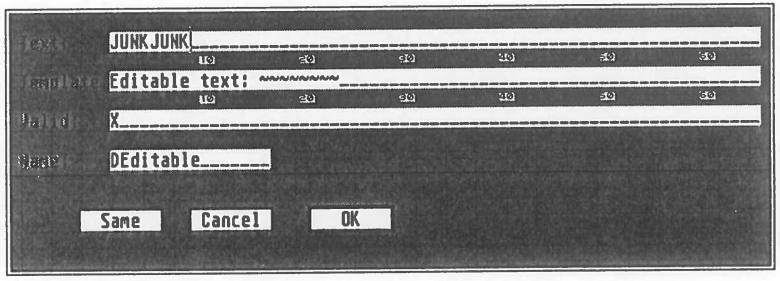
For example a Box only has a Name and presents a dialog box like this:



Buttons have one Text field and so have this type of box:



Whereas FText and FBoxText items have the appropriate TEDINFO fields as well:



To make the Name the same as the Text, click on the Same button - this is like clicking on OK except that the object will be given a name based on the text of that object and the Prefix for this tree, if any. This can save a great amount of tedious object naming. For editable text the name is taken from the Template thus giving the same name as your prompt.

Since underline characters are used by WERCS in a special way then, when editing text fields, you should enter underline characters (_) as tildes (~) and vice versa. If we didn't do this it would be impossible to see how many underlines you have in your Template strings.

To enter control characters into strings enter \\ (two back slashes) followed by the ASCII symbol corresponding to the control character. For example the ALT key symbol is entered as \\7. Similarly the copyright symbol (©) is \\189. Don't try and enter a null character \\0 as the AES treats this as the terminator of the string.

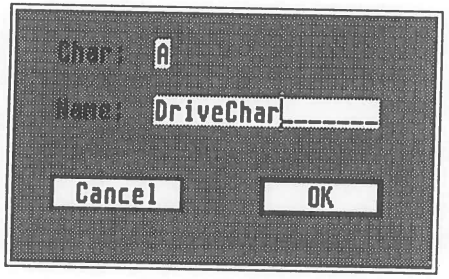
In the unlikely event that you need to enter two consecutive back-slashes type three instead; for three back-slashes type four and so on.

When using formatted text (FText or FBoxText) you should ensure that the Text field has the same number of characters as the Template field has ~s (stored in the file as underlines). If you are using different Valid characters then you should have the same number of characters in the Valid field as there are ~s in the Template field.

If you are using the same character throughout the Valid string you can enter just one as in the example above. We have not seen this facility officially documented but it works with all known versions of the operating system at the time of writing.

The other attributes of TEDINFOs (such as Large/Small characters) are set using the Text menu.

BoxChars (single characters surrounded by boxes) have their own dialog box, thus:



Moving and Sizing Objects

To change the size of an item, place the mouse near its bottom right-hand corner and drag to the required size. By *near its bottom corner* we normally mean within *one character cell* of the bottom corner but inside the object. If the object is less than a character high then you should click within the bottom half-character of the box. Similarly if it is less than one character wide you need to click within a half-character of the right.

Note that the border of an object is often outside the object itself as is any outline or shadow of a box. This area is not considered part of the object by the objc_find call. This means that you will not be able to select or drag an object by clicking in its border, outline or shadow. Also, any program you write to handle such objects must bear this in mind.

To move an object within a tree, drag from somewhere other than the bottom right-hand corner. If the object has any children, they will move with the object.

To move or size any parent or siblings of an object, first select the required objects using ALT-clicking as described under **Selecting Objects** above and then drag as if you were moving a single item.

If you want a quick copy of an object or objects, select and then Shift-drag. Generally it is best to Shift-click with the mouse near the top left corner of the object as this is where the object will appear. This will let you move a new copy of the object leaving the old one where it was, so you can drag the new one to its new position. This is particularly useful if you have a set of similar objects with the same flags and attributes set (say disabled, right justified small Text). Set up the first one and then Shift-drag to create the rest.

If you using Shift-clicking to move an object which has children, you will get copies of the children too. When you let go of the mouse you will be asked whether you wish to delete the children too. Indicate that you do wish to delete the children; otherwise you will get an extra set of children, starting where you originally clicked.

If you move an object outside its parent then you will be asked for confirmation of this (unless you are in Expert Mode).

If you move an object so that it would completely cover another object or objects then you will again be asked if you wish to adopt these objects as children of the object that you have moved. If the new position of the object will partially cover another then you will be given an error message.

After you have moved or sized an object it may be snapped to the nearest character or half-character boundary, if you have used the Auto Snap or Half Character Snap commands.

You may also change the position and size of an object using the Extras command from the Flags menu.

Editing Images

Double-clicking on Images will bring up the Icon Editor which will give a screen display similar that below:

The largest and main part of the display is used for editing the Image a pixel at a time. Beneath this is an Actual Size representation of the Image, as it will appear in your form and to the right are various buttons that you may click on.

To change an individual pixel, just click in the appropriate place on the screen; if it was black it will become white and vice-versa. To make a number of pixels the same colour click and drag; note that the actual size display will only the updated when you release the mouse button.

At the top of the button area are the buttons for changing the *height* of the Image together with the current height (28 in the example above). To increase the height click on H+ and to decrease it click on H-. Both of these work one pixel at a time and will repeat if you hold the mouse button down. The size of the main display changes to ensure that it is as large as possible whilst still displaying the Image at actual size beneath it.

If you decrease the height by too much, increase the height again and the newly displayed area will be the same as it was before you made the Image display smaller. The maximum height of Image that you can edit is 128 pixels. If you attempt to edit a larger image, it will be truncated to 128 pixels high.

To change the *width* of the Image click on the W+ and W- buttons; the current width is displayed to the right of these buttons. GEM restricts the size of Images to multiples of 16 pixels (so that it can draw them on the screen quickly) so these buttons change the width 16 pixels at a time. The width of the main display and the button will change to give as large a main area as possible. These buttons do not repeat if you hold them down. As with height changes, do not worry if you make the width too small by mistake, just click on W+ and the area that you have just deleted will re-appear.

The maximum width of an Image that can be edited is 128 pixels. Again, editing an image that is more than 128 pixels wide will cause it to be truncated.

The arrow buttons scroll the main display in the appropriate direction. Scrolling upwards and to the left loses the pixels that are removed from the Image. The pixels that are lost when scrolling to the right or downwards can be retrieved by scrolling to the left and upwards, assuming that the maximum size of 128x128 is not reached.

The Clear button will clear the entire Image to white; unless you are in Expert Mode you will be prompted to check that this is what you want.

The Fill button will fill the entire Image to black; as with Clear you will normally be prompted for confirmation.

VFlip and HFlip reflect the Image in a vertical/horizontal line through the middle of the Image. The best way to understand this is to try it. Clicking on VFlip (or HFlip) twice is like doing nothing at all.

Line is used to draw a line of black pixels. The mouse cursor will change to a +; click where you would like the line to start and then on where you would like the line to finish.

Cancel is used to cancel all the changes that you have made since entering the Image Editor; if you are not in Expert Mode you will be prompted to ensure that this is what you require.

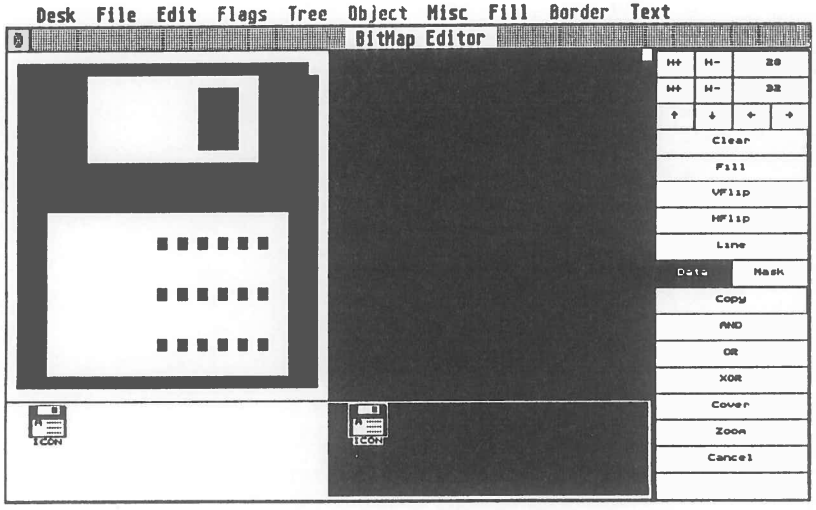
To name an Image, double-click in the Actual Size area; the usual name box will then be displayed.

The Text menu can be used to set the foreground colour of the Image.

The normal way to exit from the Image Editor is via the Close box although you can also use the commands on the File and Misc menus.

Editing Icons

The display when editing an Icon is like this:



Editing an Icon is like editing an Image except that the main display consists of the Data of the Icon on the left and the Mask of the Icon on the right. There are also some extra buttons in the Icon Area, and the Icon's string and character fields can also be accessed.

There are two Actual Size displays; the one on the left normally shows the icon not selected; that on the right shows it selected. If however you set the Selected bit using the Flags menu from the Object Level window then these will be the other way round. The Icon on the left is always as it will appear in the file.

The extra buttons for Icons are used as follows:

Data and Mask are a pair of radio buttons; if Data is selected then the Data bit map is used as the source for the commands below and also as the current bitmap for Clear, Fill, VFlip and HFlip; WERCS will remind you which bitmap you are destroying unless you are in Expert Mode. The rest of the commands are described assuming that Data has been selected; to perform the action the other way click on Mask to select it first.

Copy, AND, OR, and XOR perform the appropriate logical operation; so that Copy will make the Mask the same as the Data; AND will set only the bits in the mask that are already set in both the Mask and the Data; OR will set bits that are set in either or both and XOR will set those bits that are different in the two bitmaps.

Cover will copy the mask and surround the bits that are already set with extra bits. The source bitmap should not have any pixels set around each edge as these will be cleared in the source so that it can be covered correctly. This is useful for producing the first attempt at an Icon's mask from its Data bitmap; this is another command that is best understood by experiment. As usual you will be warned about the area that will be destroyed before proceeding if you are not in Expert Mode. If you delete something unintentionally you can always use Cancel to revert to the Icon before you entered the Icon editor.

Zoom causes the current selected bitmap, Data or Mask, to take up the whole of the main display; this is intended for editing large Icons where each pixel is very small in the main display. Click on Zoom again to return to the normal display.

Visually a finished Icon has three components; the Bitmap part, the String (ICON in the example above) and the Character (A in the example above). Every object of type Icon has an overall size just like any other GEM object; this is normally bigger than the Bitmap itself. The three components may each be positioned independently relative to the top left corner of the object. In the example, the Bitmap is to the left of the box and the Text near the bottom in the middle. Strangely the single Character's position is actually relative to the bitmap not the main object. Also GEM will draw the Bitmap and String even if they are outside the object's box.

To edit the text of the Icon's String, double-click on the text in the Actual Size display; this will bring up the usual text name box so that you can enter the String and also set the Name of the Icon object. The Icon Text may also be moved in the normal manner. Editing the Icon's Character works in a similar way and you may also move the Bitmap itself within the nominal box represented by co-ordinates of the object. The object's co-ordinates are changed as usual in the Object Level window.

Frequently Icons do not need either or both of the String and Character attributes; you can just set these to be blank. So that you can edit such text, on entry to the Icon Editor, blank strings are represented as _____ and blank characters as _. As a result of this and the fact that the actual display for icons is simulated (so that WERCS knows accurately where the components are) the Actual Display is not quite the same as the GEM display in the main Object Level window or when the Icon is displayed by your program.

We will now describe the menus in detail.

File Menu

The File menu is used to manipulate which file you are editing. Initially you are editing a blank file, shown within a window labelled Untitled.

New

To starting editing a new empty file, click on the New item from the File menu.

Loading

To load an existing resource file click on Load and select the appropriate file from the File Selector. An alert box saying .HRD file not found will appear if this file is missing - you will still be able to edit your file although any names previously attached to trees or objects will be lost.

Another way of loading a file is to set up the GEM Desktop to load WERCS when you click on .RSC or .HRD files, using Install Application. Similarly you may invoke WERCS from a CLI (such as Batcher) in which case the file extension is not required.

If you wish to load a resource file created with another resource editor you may like to convert the other editor's equivalent of the HRD file into a true .HRD file, in order to preserve the names of your items, using the conversion utility WCONVERT.

Importing Images

Images and Icons converted using the WIMAGE utility can be imported using the Import Image item on the File menu. You will be presented with the File Selector to enter the file to import. This will copy the object to the Clipboard, subsequently selecting Paste from the Edit menu will place it in your file.

This command actually copies the second object from the first tree in the file to the Clipboard.

Saving

Clicking on Save As will present you with the standard file selector and you can choose a filename for the current file. Clicking Save saves the file, without pause, under its original name - if the file was Untitled then Save will do a Save As.

The filename you enter into the File Selector for Save As need not have any extension - suitable extensions will be added by WERCS. In addition to a .RSC file, an .HRD (for HiSoft Resource Definition) file is also saved. This is a special file which contains such details as the names you have selected for the contents of that file and which other language files are to be created. This method ensures that, once you decide a particular resource file is to be used for C, for example, WERCS will know each time you edit it. The .HRD file format is described in detail in **Appendix B**.

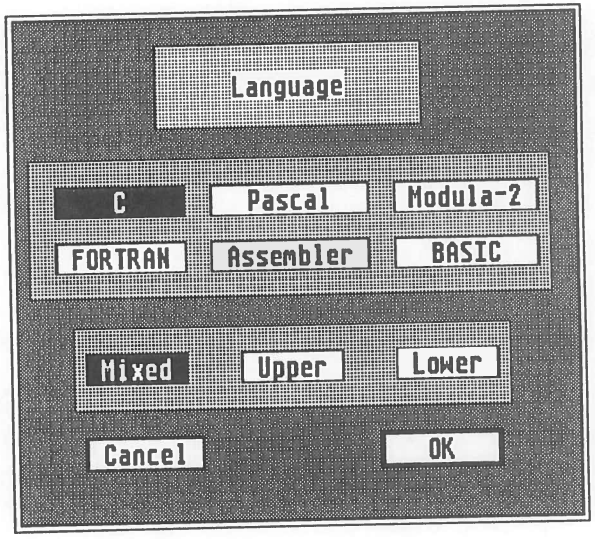
Save Prefs

The default values for various options such as the language that you are using and the character snap for new files when WERCS is loaded are read in from a file called WERCS.INF. This is searched for on the standard GEM path.

To change the defaults use the Save Prefs item on the File menu.

Language

To change the files that are created when you use Save, click on the Language item on the File menu. This will present you with the following dialog box:



You can select the language used for the header file when you next Save the resource file. It also allows you to select the names to be in lower-, upper- or mixed-case. Details of supported languages and file extensions may be found in **Appendix B**.

Quit

To leave WERCS, click on Quit. If you have changed the file you are editing you will be given the opportunity to save or lose your modifications. You can also use the Close box on the tree window to achieve the same effect.

Flags Menu

This menu contains the various attributes that are part of the ob_state and ob_flags fields of object tree items. A selected item is shown ticked. The corresponding standard GEM names for the fields are as follows :

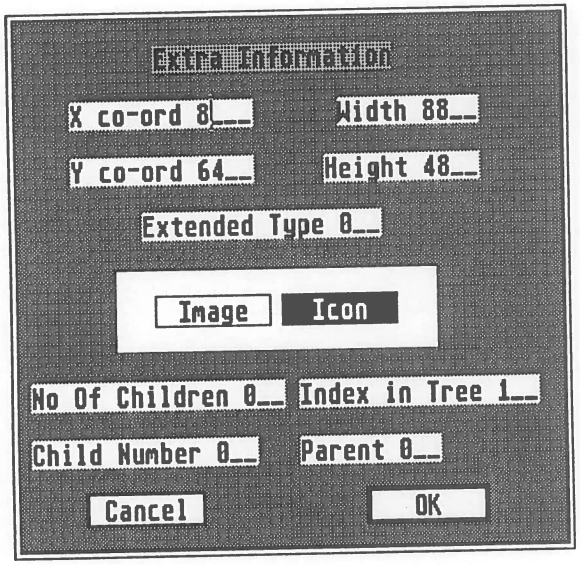
Selectable	SELECTABLE
Default	DEFAULT
Exit	EXIT
Editable	EDITABLE
Radio Button	RBUTTON
Touch Exit	TOUCHEXIT
Hide	HIDETREE
Selected	SELECTED
Crossed	CROSSED
Checked	CHECKED
Disabled	DISABLED
Outlined	OUTLINED
Shadowed	SHADOWED

UnHide Children

The item Unhide children will clear the HIDETREE bit for any immediate children of the selected object so that they become visible and you may then select them once more.

Extras

This displays a dialog box similar to that shown below which allows direct access to the object's internal structure and thus care should be taken when using this command.



The X, Y, Width and Height items are relative to the object's parent in pixels. These may be modified; use this with care as you can easily make objects move outside their parents.

The Extended Type is the most significant byte of the object word. This is ignored by the AES but may be used for your own purposes.

No Of Children is the number of first generation children that an object has. It does not include 'grand-children'.

Index in tree is the object number relative to the root object of the tree.

Child number is 0 for the first child of its parent, 1 for the second and so on. This field may be changed, in which case the objects between the old and new positions will change position in the tree. The easiest way to place the children of a particular parent in a particular order is to select the first child and make this child 0 then select the second child and make this child 1, etc.

Objects may also be re-ordered using Sort from the Misc menu.

Parent gives the Index In tree number of the object's parent.

The buttons in the box (Image and Icon in the above example) tell you what type the selected object is and let you change the object's type. Image may be changed to Icon, Box to IBox, String to Button, and Text, FText, BoxText and FBoxText interchanged.

Changing Icons into Images loses the mask and string items of the Icon so you are prompted for confirmation unless you are in Expert Mode.

Fill Menu

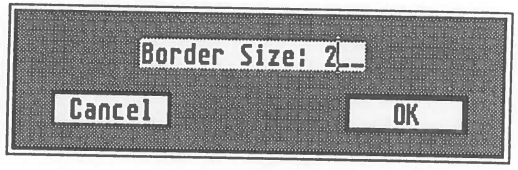
The Fill menu lets you change the fill pattern of the object, the colour of the fill and whether it is *opaque* or *transparent*. Opaque means that text will be displayed with a white background whereas transparent means that the fill pattern and fill will show 'behind' the text.

The fill pattern and colour are only applicable to Box, BoxChar, BoxText and FBoxText objects. The transparent/opaque setting is only applicable to BoxText, Text, FBoxText and FText objects.

The Fill menu is also used to set the background colours of icons.

Border Menu

Lets you change the colour and size of the border for an object. Clicking on the Size item brings up a box as below:



The Border Size is specified in pixels as appropriate. A negative size means the border is drawn inside the box; a positive number means it is drawn outside. This is only normally useful with Box, BoxChar, BoxText, FBoxText and IBox objects.

If set for FText or Text objects, the border affects the size of the box that is drawn when the object is selected thus increasing or decreasing the visual size of the object.

Text Menu

The Text menu lets you change the justification, colour, and size of the text object types: BoxChar, BoxText, FBoxText, Text and FText. The actual text is changed by double-clicking on the object.

Clipboard

The clipboard is a special area of memory which can contain trees or objects. It is ideal for moving or copying items between different areas of a resource file, or between different resource files. All clipboard commands can be found on the Edit menu.

Cut

Cut will copy the currently-selected object to the clipboard with its children and removes the current object from the tree. If the object has children you will be asked if you wish to delete them as well. If you choose not to delete the children they will become the children of the deleted object's parent. The object may then be pasted somewhere else.

Paste

Changes the mouse form to a pointing finger and waits for you to left-click. This will place a copy of the object at that position. To cancel this, click on Cancel on the Edit menu.

Copy

Copy copies the current selection to the clipboard and leaves it in place.

Cancel

Cancel is used to cancel the selection of a menu when the mouse has changed to a non-pointer form. For example, if you click on String from the Object menu and decide that you do not want a new string after all, click on Cancel.

From within the Image/Icon editor, Cancel will cancel all the changes you have made since you entered the Image/Icon editor. You are prompted with a dialog box first to ensure that this is really what you wish to do.

Abandon Edit

This allows you to abort the object-level editing that you are currently performing. All changes made since you chose to edit the current tree will be lost. It's ideal if you have made a major mistake in editing a particular tree.

Delete

Delete works like Cut but leaves the contents of the clipboard intact.

Note

To copy just some of the objects of a parent, create a new Box using the Object menu and make this cover the objects you wish to copy. Then select Copy and use Delete (*not* Cut) to delete your temporary Box, but not its children. Now use Paste to place the copy of the objects where you need them and then Delete to remove the outer Box again. This sounds more complicated than it is in practice.

Misc Menu

Auto Size

With Auto Size enabled, every time you change the text of an object the size of the object's box will change to just surround it; thus if you make the text of a Button longer it will make the Button bigger; shortening the string will make the box smaller. If you switch Auto Size off, the Button would stay the same size and the new text would not necessarily fit in the existing box.

Auto Naming

If Auto Naming is enabled (shown by a tick) then objects are automatically given a Name as if the Same button in the Text dialog box had been clicked. The Name is based on the tree's prefix, if any, and the Text of the item.

Auto Snap

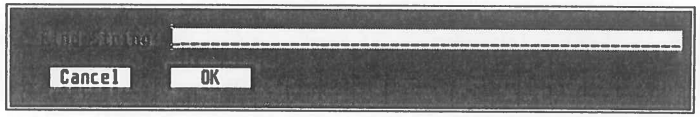
If this item is selected from the Misc menu then every item that you move or size will be *snapped* to the nearest character boundary. This is useful to make sure that items line up and will appear the same in different screen resolutions.

Half Char Snap

If this item is selected from the Misc menu then objects will snap to the nearest half-character boundary, in a similar way to character snap. However if you are designing a resource file to run in more than one resolution then objects will not necessarily come out the same, as half a character in one resolution may be either a whole character or a quarter of a character in another resolution.

Find Text

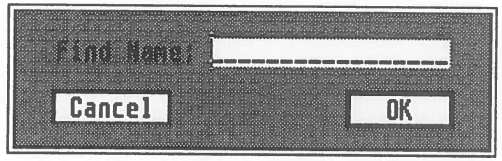
This enables you to find occurrences of a particular string within the text fields of the objects. You are presented with a dialog box, as below,



For example, if you have a number of menus and cannot remember which menu contained the item Stop you could use this command. The appropriate tree is opened and the object containing the string is selected.

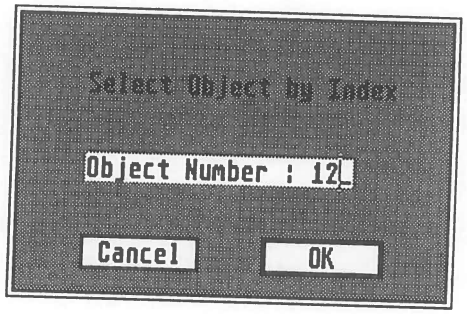
Find Name

This item searches for a particular named object within the file, opens the appropriate tree and selects the object. The box presented looks like this:



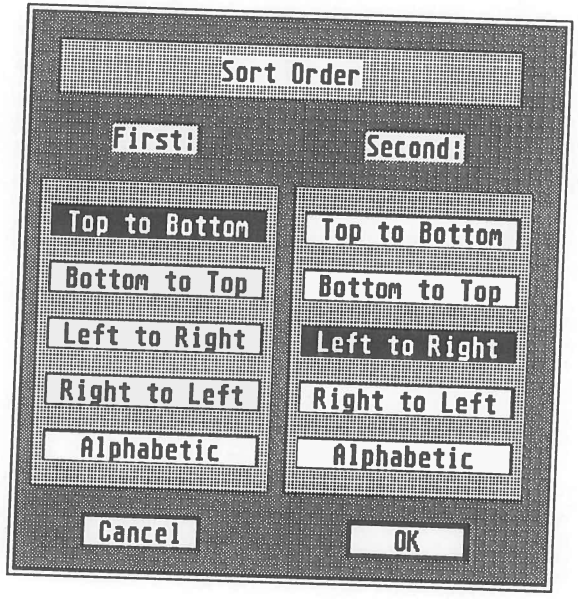
Number Select

This allows you to select an object given its *object number* in the current tree; this can be useful if you have an object outside its parent.



Sort

This enables you to sort the children of a particular object according to various possible criteria that are selected from the dialog box:



Top to Bottom and Bottom to Top will sort the objects according to their y position on the screen whilst Left to Right and Right to Left will sort them according to their x position on the screen. There are two priorities for the sort, First and Second. Note that the sort *does not* affect the objects' positions on the screen, it affects their order *in memory* and *within* the tree.

The default is First, Top to Bottom and Second, Left to Right. So, say we have 6 objects (names objA to objF in any order in the tree and in memory) with screen representations as follows:

objd objA objE

objB objF objC

and then we sort using the default options. The objects will be sorted so that their order in memory and in the tree is objd, objA, objE, objB, objF, objC. Note that the sort will *not* affect the screen representations.

Alphabetic means that the *strings* of the objects are compared rather than their screen positions. Sorting alphabetically does *not* mean that the objects will change position on screen only that their position in the object tree and in memory may change.

Remember to select the parent of the objects that you wish to sort before you click on Sort. You can use Alt-clicking to select the parent of a given object.

Test

Test lets you test out a Form, Menu or Alert Box. In order to test a Form it must have an object (such as a Button) with the EXIT and SELECTABLE flags set, or alternatively with the TOUCHEXIT flag set. If it does not then you are given an error message.

When you click on an Exit Button (or click on an item if testing a Menu) then you are told which item you have selected and its name, if any. You can choose whether to continue testing the tree, or return to WERCS. If you double-click on a TOUCHEXIT item then the value displayed will not include the top bit, as returned by form_do.

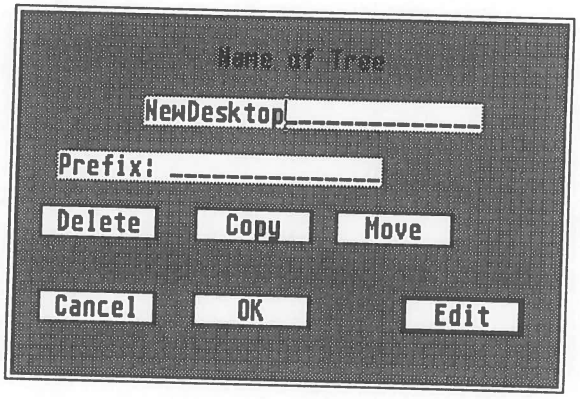
Expert level

If this is enabled (shown by a tick) then all warnings to do with tree re-organising are suppressed: for example, when an object is given a new parent, or commands that effect the entire data or mask bitmaps in the icon editor. This also includes the warning about losing information when changing an Icon to an Image using Extras and when using the Abandon Edit command.

Tree Level Editing

Forms

Forms are the most common type of tree in resource files; they are normally used for dialog boxes, but can also be used for replacement desktops, to change the pattern of the background in a GEM program or to add icons to it. When you create a new Form or single-click on an existing one in the file window, you are presented with a dialog box like the one below:



The name of the tree may be changed. WERCS will check to ensure that it is a valid name according to the current selection of Language, with the correct case, and that it is not a duplicate of a current name. If the other item with this duplicate name is an object you can use the Find Name command to select it and then change its name.

Remember that in Mixed case Ok and OK are different names but if you have selected Lower or Upper case then they are not.

Pressing the Return key or clicking on the Edit button will then let you edit the objects within the tree.

Cancel will not add a new tree to the file and will disregard any changes to the tree name that you have made.

To add more than one tree without editing them immediately, click on the OK button - this lets you set up a number of trees without entering the objects.

To re-order the trees in a file, click on the Move button. This will change the mouse form to a pointing finger; you should then click on the tree that you wish to place immediately *after* the current form. Owing to the structure of resource files, when the file is reloaded, the Menus and Forms will be first, followed by the Free Strings and Alerts, followed by the Free Images.

To delete or copy an entire tree, click on the appropriate button. To paste a tree from the clipboard into your file, click on Paste from the Edit menu.

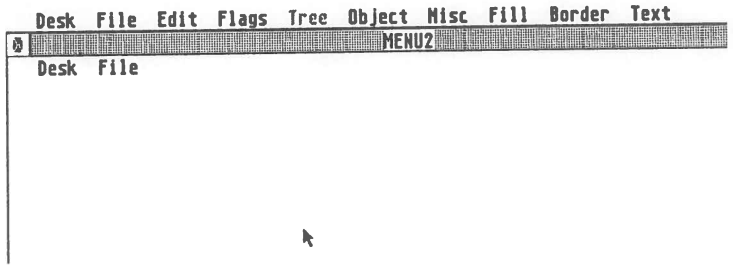
This box also lets you set up the Prefix for this particular tree. This is used to provide the start of the names of objects if you use Auto Naming.

To edit an existing tree, double-click on the appropriate tree icon. You will then be taken straight to the Object Level.

Menus

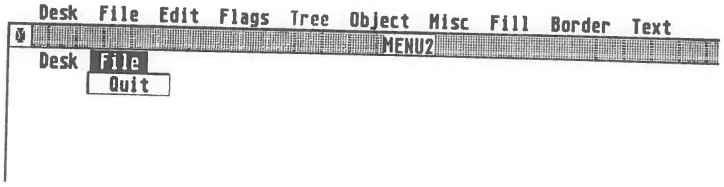
Menus are a very special type of Form which must conform to a number of unpublished rules, otherwise GEM will behave strangely. Fortunately, when using WERCS, you don't have to worry about these rules as WERCS will cope with them for you.

When you ask for a new menu you will see a screen similar to that below:



Normally Menus consist of Titles (which are displayed along the top of the screen) and Strings (which are displayed in the pull-down menus themselves). To add a new Title, click on Title from the Object menu and then click in the menu bar where you would like it. The other Titles (and their menus) will be moved if required.

To add items to a given Title, first click on the Title itself; this will cause the appropriate Menu to appear, for example:



You can then insert objects in the usual manner, normally Strings, and objects below the new object will be moved down. You should ensure that the mouse pointer is at the left edge of the box when inserting objects; otherwise you will leave a 'hole' to the left of it.

You can use types other than Strings in Menus if you wish; we used WERCS to produce its own resource file, for example. You can also change the flags and states of items just as if you were editing a Form. For those objects that have them, the items on the Fill, Border and Text menus can be used.

If you want your menu to work in more than one resolution don't use Icons or Images or you will find that there will either be gaps between them or they will overlap. This is because the width and height of these objects are a different number of character cells in different resolutions. This can be avoided by adjusting the object once it is loaded so that there are no gaps between icons.

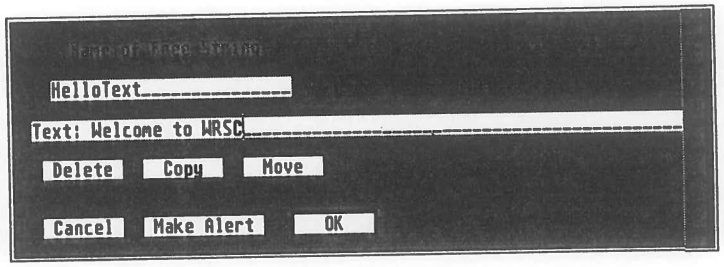
The boxes surrounding menus must not be take up more than one quarter of the screen, otherwise the system may crash. Be especially careful, when designing menus for use in Low Resolution.

The Tree Name box can be used in the same way as for Forms.

Free Strings

A Free String is a string of characters that is not connected with any particular tree. They can be used to facilitate foreign-language versions of software, for example.

The Name and Text of the Free String can be modified in the same way as any other type of string in WERCS so that you can use \\ to enter control and graphics characters for example.



To edit an existing Free String, it is only necessary to single- or double-click to bring up the box shown above.

The Delete, Copy, Move and OK buttons work in the same way as with Forms, detailed earlier.

Make Alert can be used to turn a Free String into an Alert. You should ensure that the String conforms to the rules for Alerts, as described below.

Alerts

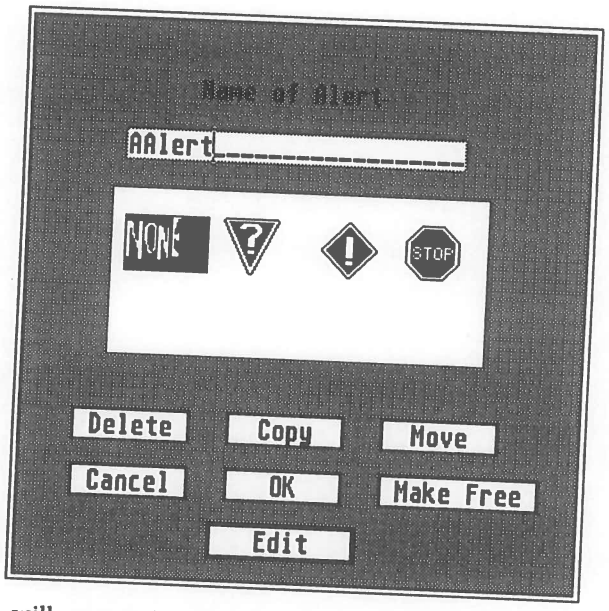
Alert Boxes are actually stored as Free Strings (see above) but are passed to the AES form_alert call to display an alert box.

There are two types of restrictions as to the contents of Alerts; the first type is those restrictions documented by Atari (to keep down the amount of memory used by the AES when displaying them) and the second type of restriction is caused by bugs in the first release of the operating system ROMs.

As officially documented, each line in an alert box must be no more than 30 characters and there is a maximum of 5 lines. Each Button must be no more than 20 characters each and there is a maximum of three Buttons. Strings and Buttons may not contain] or | characters. ROMs prior to 1.2 do not check for the infringement of these rules and failing to adhere to them will corrupt certain areas of the AES workspace! WERCS rigidly enforces these rules when converting the tree representation back to a string.

The release 1.0 of the ST ROMs contained various bugs, including long buttons/short text problems. Before releasing a commercial program ensure you have checked all your Alerts on a 1.0 ROM machine. Subsequent ROM releases (1.2 a.k.a. Blitter TOS, 1.4 a.k.a. Rainbow TOS and 1.6 a.k.a. STE TOS) have these problems corrected. These difficulties with the early ROMs have not been documented and so WERCS cannot check reliably for them. If you have later ROMs you have the flexibility to use some Alerts that it is not possible to use with the earlier ROMs.

The dialog box that you are presented with, when you single-click on an Alert Box in the file window, looks like this:



Which icon will appear in the Alert Box is controlled by clicking on the appropriate icon in the tree display as above.

The Delete, Copy, Cancel, OK and Move buttons work in a similar way to those on the Form Tree Name dialog box.

Clicking on Make String turns this item into a Free String rather than an Alert.

Clicking on Edit (or double-clicking on the the icon from the tree level display) will open an Object Level window that looks similar to that opened when you are editing a Form.

You should only add Strings and Buttons to the Form and these will be re-positioned automatically by WERCS. You can edit the Text in the normal way and also delete, copy and paste objects. Modifying the flags and states of the parts of an object will not affect the final Alert Box.

Re-ordering the Buttons in Alert Box is achieved by dragging a Button. If you drag button A onto Button B then the Buttons will be re-arranged so that Button A is immediately before Button B. This is similar to the moving of Titles in Menus. If it sounds complicated, experiment and you should soon get the hang of it.

Alerts are represented as strings of the format shown below:

```
[icon][line1|line2 ... |linen][button1|button2....]
```

where Icon is one of:

0 No icon
1 Question Mark icon
2 Exclamation Mark icon
3 Stop icon

line1, line2 etc. are the various message lines and button1 are the various Buttons. To check that you understand this, create an Alert and then change it to a Free String and, assuming that it is small enough to fit on the screen, you will be able to inspect it.

Free Images

A Free Image is an Image-type object that is not connected with any particular tree. When you use rsrc_gaddr you get the address of a BITBLK rather than an object.

The Tree Name dialog box for Free Images works in the same way as that for Forms except that clicking on Edit takes you straight to the Image Editor as for Image objects.

Keyboard Shortcut Summary

The following table gives the keyboard shortcuts when the Alt, Ctrl or Shift keys are held down.

Key	Alt	Ctrl	Shift
A	Abandon Edit	Border Size	Alert
B		Shadowed	Box
C	Copy	Crossed	BoxChar
D		Default	Form
E	Extras	Editable	Button
F	Find Text		FText
G	Find Name	Disabled	FBoxText
H	Selected Number	Delete	
I	Import Image	Small text	IBox
J		Large Text	Icon
K	Expert	Hide	Image
L	Load	Left	Free Image
M		Centre	Menu
N	New	Right	
O	Sort	Outlined	
P	Language	Opaque	ProgDef
Q	Quit	Transparent	
R	Save As	Radio Button	Free String
S	Save	Selectable	String
T	Test	Touch Exit	Text
U	Auto Naming	Un Hide	BoxText
V	Paste	Selected	Title
W	Auto Size		
X	Cut	Exit	
Y	Char Snap	Checked	
Z	Half Snap		

Note also that the Backspace key is used to delete objects, whilst the Undo key cancels an operation.

There is a rationale behind the choice of keyboard shortcuts to help you remember them; the Alt keys refer to commands on the File, Edit and Misc menus, Ctrl for the Flags, Fill, Border and Text menus and Shift for the Object and Tree menus. We have attempted to make shortcuts use the initial letter of the item as far as possible; the exceptions to this are the standard clipboard shortcuts and the following:

- P Programming Language,
- O Order (Sort),
- ^G Greyed (Disabled),
- ^B Border (Shadowed).

MonST2C
The Debugger

Introduction

MonST was originally designed as a low level debugger for debugging assembly language programs but we, and many other people, have found it useful in debugging C programs. The version of MonST that we supply with Lattice C, MonST2C, has been enhanced so that it 'knows' about where, in memory your program lines start and automatically loads your program and source code for you.

Together with the facilities of the original MonST, such as function labels, a full integer expression evaluator and named access to external variables, MonST2C gives you many of the features of a high level symbolic debugger. It does not give you access to local variables, floating point numbers, structures etc. As it was originally designed as a low level debugger some knowledge of assembly language is useful when using MonST2C.

As MonST2C uses its own screen memory, the display of your program is not destroyed when you single-step or breakpoint, making it particularly useful for graphical-output programs such as GEM applications. It also uses its own screen drivers so it is possible to single-step into the operating system screen routines such as the AES or BIOS without affecting the debugger. MonST2C will also work in low resolution, thus allowing you to debug programs that run in low resolution.

Initially we shall describe how to use MonST2C to debug programs that are compiled as a single module. Using it with multi-file applications will be described later.

If you are already an expert at using the version of MonST supplied with DevpacST version 2 then please read the next two pages on the use of MonST2C with Lattice C.

Preparing to use MonST2C

If you are going to debug a program using MonST2C you should ensure that you have selected the compiler's -d3 flag when compiling your program, either using the Compiler Options box or from the compiler's command line.

When linking you should either check Linker Symbols from the Options menu if invoking the linker from the editor or use the XADDSYM keyword if using an explicit linker command line or link file. Do not use the -Ln option as this will remove much of the debugging information.

This will ensure that the addresses of your functions, the library functions that you use, your program's external variables, and the address corresponding to each line of your source code will be stored in your executable file. Don't be surprised if this causes it to increase in size dramatically!

Whilst we recommend using -d3 for most purposes there are two other variants of the -d flag that you may find useful:

Using -d1 includes the name and line number information for MonST2C, producing very much smaller files than using -d3, but has two disadvantages compared with -d3.

Normally the compiler will make some of your variables register variables automatically. Using -d3 will force the compiler to store every such auto-generated register variable in memory at the end of each statement. This has the advantage of generally making the assembly language code generated easier to understand. If you are using the -d1 flag then it is probably a good idea to use the -mr compiler flag which will disable the automatic registerisation completely.

The other disadvantage of -d1 is that it only stores the filename of the source file in the debug information rather than the full pathname. Thus it can only be used to debug files that are in the current directory.

The -d2 option is exactly the same as -d3 except that the output files are slightly smaller and the compiler chosen register variables will not be flushed to memory; they will *remain* in registers.

Using -d4 and -d5 will create still larger files than -d2 and -d3, however the additional information is of no use to MonST2C.

Don't use the Global Optimiser when preparing code to be debugged with MonST2C as the transformations that it performs can make the code produced appear to bear very little similarity to your source code!

Invoking MonST2C

From the Desktop

MonST2C is supplied as a GEM program with extension .PRG; to debug a TOS application you can rename it as MONST2C.TOS or install it as a TOS program. This will ensure that the operating system will perform the same initialisation as if you were running your program without it. Once executed MonST2C will prompt you for the name of the file to load.

If you debug a TOS program with the GEM version of the debugger it will work fine but the screen display will probably be messy; however, debugging a GEM program with a TOS debugger will cause all sorts of nasty problems to occur and should be avoided.

From the Editor

If you are using the standard editor configuration file, EDCTOOLS.INF, MonST2C can be invoked using the Debug option on the Tools menu and Alt and the 5 key on the numeric key pad. Of course, the editor will need to be able to find the MONST2C.PRG file for this to work. Invoking the debugger in this way will load the .PRG file corresponding to the file being edited.The debugger will also load your source file too.

The type of initial screen mode used when invoked from the editor is determined by the GEM and TOS buttons in the Tool Configuration dialog. See the section **EdC, The Screen Editor** for more details. The rules described above about using the wrong type of screen initialisation are also relevant here.

From Batcher

If you wish to invoke MonST2C from Batcher, just type

```
monst2c test
```

if test is the program that you are debugging.

MonST2C Dialog and Alert Boxes

MonST2C makes extensive use of dialog- and alert-boxes which are similar in concept to those used by GEM programs but have several differences. MonST2C does not use genuine GEM-type boxes in order for it to remain *robust* - that is to avoid interaction when debugging programs that themselves use GEM calls. In addition the mouse is not available within the debugger itself which makes objects like true GEM buttons impossible.

A MonST2C dialog box displays the prompt ESC to abort above the top left corner of the box together with a prompt, normally followed by a blank line with a cursor. At any time a dialog box may be aborted by pressing Esc, or data may be entered by typing. The cursor, Backspace and Del keys may be used to edit entered text in the usual way and the whole line may be deleted by pressing the Clr key - note that this is different to GEM dialog boxes which use the Esc key to delete a whole line of text. An entered line is terminated by pressing the Return key, though if the line contains errors the screen will flash and the Return key will be ignored allowing correction of the data before pressing Return again. Another difference is that dialog boxes that require more than one line of data to be entered do not allow the use of the cursor up and down keys to switch between different lines - in MonST2C the lines have to be entered in order.

A MonST2C alert box is a small box displaying a message together with the prompt [Return] and is normally used to inform the user of some form of error. The box will disappear on pressing the Return or Esc keys, whichever is more convenient.

Initial Display

If you have run MonST2C without a command line you will be presented with a dialog box prompting for an executable program name. You should enter the name of the program that you wish to debug. If you omit the file's extension, MonST2C will look for a .PRG, .TTP and .TOS file in that order.

Low Res

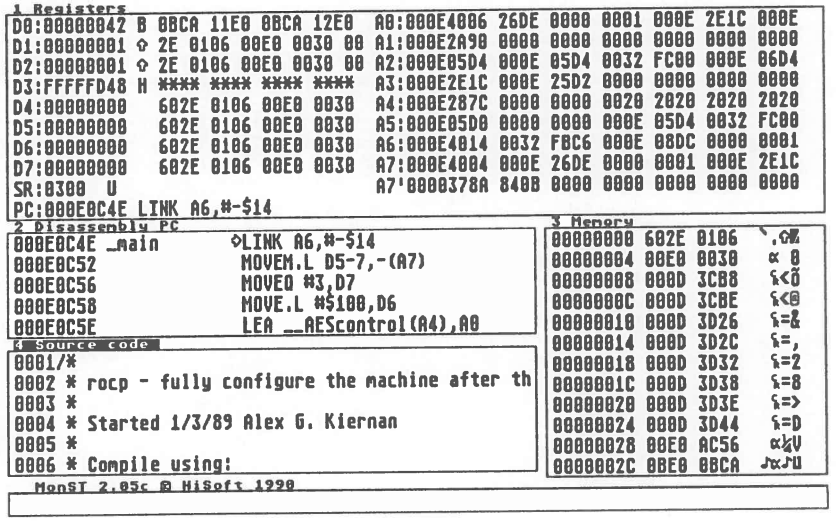
Certain features work differently or are not available when using MonST2C in low resolution. They are shown with this icon.

Front Panel Display

The main display of MonST2C is via a *Front Panel* showing registers, memory and instructions. The name Front Panel stems from the type of panels that were mounted on mainframe and mini computers to provide information on the state of the machine at a particular moment, usually through the use of flashing lights. These lights represent whether or not particular flip-flops (electronic switches) within the computer are open or closed; the flip-flops that are chosen to be shown on this panel are normally those that make up the internal registers and flags of the computer thus enabling programmers and engineers to observe what the computer is doing when running a program.

So these are *hardware* front panel displays; what MonST2C provides you with is a *software* front panel - the code within MonST2C works out the state of your computer and then displays this information on the screen.

The initial MonST2C display consists of five windows, similar to those shown below. In low-resolution the arrangement of the windows is slightly different to allow efficient use of the smaller available screen space.



The top window (1 Registers A4) displays the values of the machine's data and address registers, together with the memory pointed to by these registers.

The next window (2 Disassembly PC) is the disassembly window; this displays several lines of assembly instructions, by default based around the program counter (PC), shown in the title area of the window. A ⇒ sign is used to denote the current value of the PC.

Window number 3 is the memory window which displays a section of memory in word-aligned hex and ASCII.

Window number 4 is the source code window; it shows a a portion of your source code and the corresponding line numbers.

The final window at the bottom of the screen, which is un-numbered, is the smallest window and is used to display messages.

One of the most powerful features of MonST2C is its flexibility with windows - an extra window may be created, the font size can be changed, and windows may be locked to particular registers; these features are detailed later.

Simple Window Handling

MonST2C has the concept of a *current window* - this is denoted by displaying its title in black. The current window may be changed by pressing the Tab key to cycle between them, or by pressing the Alt key together with the window number, for example Alt-2 selects the disassembly window. (AZERTY keyboard users please note - the Shift key is *not* required when using Alt to select windows). Note that the lowest window can never be made the current window - it is used solely for displaying messages.

Command Input

MonST2C is controlled by single-key commands which creates a very fast user-interface, though this can take getting used to if you are familiar with a line-oriented command interface of another debugger. Users of HiSoft DevpacST and HiSoft debuggers on other machines should find that they are already familiar with many of the commands.

In general the Alt key is the window key - when used in conjunction with other keys it acts on the *current window*.

Commands may be entered in either upper or lower case. Those commands whose effects are potentially disastrous require the Ctrl key to be pressed in addition to a command key. The keys used were chosen to be easy to remember, wherever possible. Commands take effect immediately - there is no need to press Return - and invalid commands are simply ignored. The relevant sections of the front panel display are updated after each command so any effects can be seen immediately.

MonST2C is a powerful and sometimes complex program and we realise that it is unlikely that many users will use every single command. For this reason the remainder of the MonST2C manual is divided into two sections - the former is an introduction to the basic commands of the program, while the latter is a full reference section. It is possible for new users and beginners to use the debugger effectively while having only read the **Overview**; don't be intimidated by the **Reference** section.

MonST2C Overview

The most common low-level command in MonST2C is probably single-step, obtained by pressing Ctrl-Z (or Ctrl-Y if you find it more convenient). This will execute the instruction at the PC, the one shown in the Register window and, normally, also in the Disassembly window. After executing it the debugger re-displays the values of the registers and memory displayed, so you can watch the processor execute your program, step by step. Single-stepping is the best way of going through sections of code where you don't understand what is going on, but it is also the slowest - and it deals with your program only on an assembly language level, not in terms of your C program. There is, of course, an answer.

A *breakpoint* is a special word placed into your program to stop it running and enter MonST2C. There are many types of breakpoint but we will restrict ourselves to the simplest for now. A breakpoint may be set by pressing Alt-B, then entering the address you wish to place the breakpoint. You can enter an address in MonST2C as a symbol, as a hexadecimal line number preceded by #, hex (the default base), a decimal line number preceded by #\ or as a complex expression. Examples of valid addresses are main, foo, #10, #\123 10+mydata. If you type in an invalid address the screen will flash and allow you to correct the expression. You can omit the leading _ or @ of C function names if you wish.

Having set a breakpoint you need some way of letting your program actually run, and Ctrl-R will do this. If will execute your program using the registers displayed and starting from the PC. MonST2C will be re-entered if a breakpoint has been hit, or if a processor exception occurs.

MonST2C uses its own screen display which is independent from your programs. If you press the V key you will see your current programs display, pressing another key switches you back to MonST2C. This allows you to debug programs without disturbing their output at all.

Any window may be zoomed to the full screen size by pressing Alt-Z. To return to the main display press Alt-Z or the Esc key. The Esc key is also the best way of getting out of anything you may have invoked by accident. The Zoom command, like all Alt-commands, works on the *current window* which you can change by pressing Tab. You can dump the current window to your printer by pressing Alt-P.

To change the address from which a window displays its data, press Alt-A, then enter the new address. Note that the disassembly window will always re-display from the PC after you single-step, because it is *locked* to the PC. The locking of windows is detailed in the Reference section.

To quit MonST2C press Ctrl-C. Strange as it may sound this will not always work - what Ctrl-C does is terminate the *current program*, which may be MonST2C or, more likely, the program you are debugging., in which case MonST2C will still be in control. You know when you have terminated the program under investigation because it will say so in the lower window. Once your program has been terminated, pressing Ctrl-C will terminate MonST2C.

We hope this overview has given you a good idea of the most common features of MonST2C to let you get on with the complex process of writing and debugging programs. When you feel more confident you should try and read the **Reference** section, probably best taken, like all medicine, in small doses.

MonST2C Reference

Numeric Expressions

MonST2C has a full expression evaluator, based on that in the DevpacST assembler, GenST, including operator precedence. We decided that changing MonST2C to use the standard C operators would be confusing for users who are already familiar with MonST2.

The following operators are supported, in decreasing order of precedence:

monadic minus (-) and plus (+), address of line number (#)

bitwise not (~)

shift left (<<) and shift right (>>)

bitwise And (&), Or (!) and Xor (^)

multiply (*) and divide (/)

addition (+) and subtraction (-)

equality (=), less than (<), greater than (>), not equals (<>)

The comparison operators are signed and return 0 if false or -1 ($FFFFFFFF) if true. The shift operators take the left hand operand and shift it the number of bits specified in the right hand operand, vacated bits are filled with zeroes.

This precedence can be overridden by the use of parentheses (and). With operators of equal precedence, expressions are evaluated from left-to-right. Spaces in expressions (other than those within quotes - ASCII constants) are not allowed.

All expression evaluation is done using 32-bit signed-integer arithmetic, with no checking of overflow.

The MonST2C expression evaluator also supports indirection using the { and } symbols. Indirection may be performed on a byte, word or long basis, by following the } with a period then the required size, which defaults to long. If the pointer is invalid, either because the memory is unreadable or not an even address (if word or longword indirection is used) then the expression will not be valid.

For example, the expression

```
{data_start+10}.w
```

will return the word contents of location data_start+10, assuming data_start is even. Indirection may be nested in a similar way to ordinary parentheses.

Numbers

Absolute numbers may be in various forms:

decimal constants, e.g. \1029

hexadecimal constants, e.g. 12f or $12f

octal constants, e.g. @730

binary constants, e.g. %1100010

character constants, e.g. 'X'

\ is used to denote decimal numbers, $ is used to denote hexadecimal numbers (the default), % for binary numbers, @ for octal numbers and single ' or double " quotes for character constants.

Character Constants

Whichever quote is used to mark the start of a string must also be used to denote its end and quotes themselves may be used in strings delimited with the same quote character by having it occur twice. Character constants can be up to 4 characters in length and evaluate to right-justified longs with null-padding if required. For example, here are some character constants and their ASCII and hex values:

```
"Q"         Q       $00000051
'hi'        hi      $00006869
"Test"      test    $54657374
"it's"      it's    $6974277C
'it''s'     it's    $6974277C
```

Symbols and Registers

Symbols may be referred to and are normally case-sensitive and significant to either 8 or 22 characters (depending on the form of debug information used), though this can be changed with Preferences.

Normally there is no need to type the initial _ or @ of C language labels. You can enforce the need to include the _ or @ with the Preferences command

Registers may be referred to simply by name, such as A3 or D7 (case insensitive), but this clashes with hex numbers. To obtain such hex numbers precede them with either a leading zero or a $ sign. A7 refers to the *user* stack pointer.

There are several reserved symbols which are case insensitive, namely TEXT, DATA, BSS, END, SP, SR, and SSP. END refers to one byte past the end of the BSS section and SP refers to either the user- or supervisor-stack, depending on the current value of the status register. Remember that the names of all your external variables and functions will also be available.

In addition there are 10 memories numbered M0 through M9, which are treated in a similar way to registers and can be assigned to using the Register Set command. Memories 2 through 5 inclusive refer to the current start address of the relevant window and modifying them will change the start address of that window.

Window Types

There are four window types and the exact contents of these windows and how they are displayed is detailed below. The allowed types of windows are shown in the table below.

Window	Allowed Types
1	Register
2	Disassembly
3	Memory
4	Disassembly, Memory or Source code
5	Memory

Register Window Display

The data registers are shown in hex, together with the ASCII display of their low byte and then a hex display of the eight bytes they point to in memory. The address registers are also shown in hex, together with a hex display of 12 bytes. As with all hex displays in MonST2C this is word-aligned, with non-readable memory displayed as **.

The status register is shown in hex and in flag form, additionally with U or S denoting user- or supervisor-modes. A7' denotes the supervisor stack pointer, displayed in a similar way to the other address registers.

The PC value is shown together with a disassembly of the current instruction. Where this involves one or more effective addresses these are shown in hex, together with a suitably-sized display of the memory they point to.

For example, the display

```
TST.W $12A(A3) ;00001FAE 0F01
```

signifies that the value of $12A plus register A3 is $1FAE, and that the word memory pointed to by this is $0F01. A more complex example is the display

```
MOVE.W $12A(A3),-(SP) ;00001FAE 0F01 ⇒0002AC08 FFFF
```

The source addressing mode is as before but the destination address is $2AC08, presently containing $FFFF. Note that this display is always of a suitable size (MOVEM data being displayed as a quad-word) and when pre-decrement addressing is used this is included in the address calculations.

Low Res

No hex data is shown for the data registers and the address register data area is reduced to 4 bytes. In addition the disassembly line may not be long enough to display complex addressing modes such as the second example above.

Disassembly Window Display

Disassembly windows display memory as disassembled instructions. On the left the hex address is shown, followed by any symbol, then the disassembly itself. The current value of the PC is denoted with ⇒.

If the instruction has a breakpoint placed on it this is shown using square brackets ([ ]) afterwards, the contents of which depend on the type of breakpoint. For stop breakpoints this will be the number of times left for this instruction to execute, for conditional breakpoints this will be a ? followed by the beginning of the conditional expression, for count breakpoints this will be a = sign followed by the current count, and for permanent breakpoints a * is shown.

The exact format of the disassembled op-codes is Motorola standard, as accepted by the assembler, asm. All output is upper-case (except lower-case labels) and all numeric output is hex, except trap numbers. Leading zeroes are suppressed and the $ hex delimiter is not shown on numbers less than 10. Where relevant numerics are shown signed. The only deviation from Motorola standard is the register lists shown in MOVEM instructions - in order to save display space the type of the second register in a range is abbreviated, for example

```
MOVEM.L d0-d3/a0-a2,-(sp)
```

will be disassembled as

```
MOVEM.L d0-3/a0-2,-(sp)
```

Low Res

Any displayed symbols replace the hex address display, limited to a maximum of 8 characters.

Memory Window Display

Memory windows display memory in the form of a hex address, word-aligned hex display and ASCII. Unreadable memory locations are denoted by **. The number of bytes shown is calculated from the window width, up to a maximum of 16 bytes per line.

Source-code Window Display

The source-code window displays ASCII files in a similar way to a screen editor. The default tab setting is 8 though this can be toggled to 4 with the Edit Window command.

Window Commands

The Alt key is generally used for controlling windows, and when used applies to the *current window*. This is denoted by having an inverse title and can be changed by pressing Tab, or Alt and the window number.

Most window commands work in any window, zoomed or not, though when it does not make sense to do something the command is ignored.

Alt-A — Set Address

This sets the starting address of a memory or disassembly window.

Alt-B — Set Breakpoint

Allows the setting of any type of breakpoint, described later under **Breakpoints**.

Alt-E — Edit Window

On a memory window this lets you edit memory in hex or ASCII. Hex editing can be accomplished using keys 1-9, A-F, together with the cursor keys. Pressing Tab switches between hex & ASCII, ASCII editing takes each keypress and writes it to memory. The cursor keys can be used to move about memory. To leave edit mode press the Esc key.

On a register window this is the same as Alt-R, Register Set, described shortly.

On a source-code window this toggles the tab setting between 4 and 8.

Alt-F — Font size

This changes the font size in a window. In high resolution 16 and 8 pixel high fonts are used, in colour 8 and 6 pixel high fonts are used. This allows a greater number of lines to be displayed, assuming your monitor can cope.

Changing the font size on the register window causes the position of windows 2 and 3 to be re-calculated to fill the available space.

Alt-L — Lock Window

This allows disassembly and memory windows to be locked to a particular register. After any exception the start address of the window is re-calculated, according to the locked register.

To unlock a window simply enter a blank string.

By default window 2 is locked to the PC. You can lock windows to each other by specifying a lock to a memory window, such as M2.

Alt-O — Show Other

This prompts for an expression and displays it in hex, decimal and as a symbol if relevant.

Alt-P — Printer Dump

Dumps the current window contents onto the printer. The print can be aborted by pressing Esc.

Alt-R — Register Set

Allows any register to be set to a value, by specifying the register, an equals sign, then its new value. It can also be used to set the values of the MonST2C memories M0 to M9. For example the line

```
a3=a2+4
```

sets register A3 to be A2 plus 4. You can also use this to set the start address of windows when in zoom mode so that on exit from zoom mode the relevant window starts at the required address.

Note — Do not assign to M4 if window 4 is currently a source-code window.

Alt-S — Split windows

This either splits window 2 into window 2 and window 4, or splits window 3 into window 3 and window 5. Each new window is independent from its creator. Pressing Alt-S again will unsplit the window.

Low Res — This command has no effect.

Alt-T — Change Type

This only works on window 4 (created either by splitting window 2 or by loading a source file). It changes the type of the window between disassembly, memory and source-code (if a file has been loaded).

Alt-Z — Zoom Window

This zooms the current window to be full size. Other Alt commands are still available and normal size can be achieved by pressing Esc or Alt-Z again.

Note — Zooming the register window can be extremely useful as it allows you to see the 'hidden' registers M0, M1 and M6-M9.

Cursor Keys

The cursor keys can be used on the current window, and their action depends on the window type.

On a memory window all four cursor keys change the current address, and Shift ↑ and Shift ↓ move a page in either direction.

On a disassembly window ↑ and ↓ change the start address on an instruction basis, ← and → change the address on a word basis.

On a source-code window ↑ and ↓ change the display on a line basis, and Shift ↑ and Shift ↓ on a page basis.

Screen Switching

MonST2C uses its own screen display and drivers to prevent interference with a program's own screen output. To prevent flicker caused by excessive screen switching when single-stepping the screen display is only switched to the program's after 20 milliseconds, producing a flicker-free display while in the debugger. In addition the debugger display can have a different screen resolution to your program's if using a colour monitor.

V — View Other Screen

This flips the screen to that of the programs, any key returns to the MonST2C display.

Ctrl-O — Other Screen Mode

This changes the screen mode of MonST's display between low and medium resolution. It re-initialises window font sizes and window positions to that of the initial display. This will not effect the screen mode of the program being debugged.

This command is ignored on a monochrome monitor.

As MonST2C has its own idea of where the screen is, what mode it is in and what palette to use you can use MonST2C to actually look at the screen memory in use by your program, ideal for low-level graphics programs.

Note

If your program changes screen position or resolution, via the XBIOS or the hardware registers, it is important that you temporarily disable screen switching using Preferences while executing such code; otherwise MonST2C will not notice the new attributes of your program's screen.

When a disk is accessed, when loading or saving, the screen display will probably switch to the program's during the operation. This is in case a disk error occurs, such as a write-protection violation or a read error, as it allows any GEM alert boxes to be seen and acted upon.

Breaking into Programs

Shift-Alt-Help — Interrupt Program

While a program is running it can be interrupted by pressing this key combination, which will cause a trace exception at the current value of the PC. With computationally intensive program sections this will be within the program itself but with a program making extensive use of the ROM, such as the GEMDOS or AES, the interruption will normally be in the ROM itself, or the line-F handler stored in low-memory. If this is the case it is recommended that a breakpoint be placed in your actual program area then a Return to Program command (Ctrl-R) issued.

Pressing Alt-Help without the Shift key will normally produce a screen dump to the printer - if you press this accidentally it should be pressed again to cancel the dump.

It is possible for this key combination to be ignored when pressed - if this occurs press it again and it should work. Pressing it when in MonST2C itself will produce no effect.

Note

A program should never be terminated (using Ctrl-C) if it has just been interrupted in the middle of a ROM routine. This is likely to cause a system crash.

Breakpoints

Breakpoints allow you to stop the execution of your program at specified points within it. MonST2C allows up to eight simultaneous breakpoints, each of which may be one of five types. When a breakpoint is hit MonST2C is entered and then decides whether or not to halt execution of your program, entering the front panel display, or continue; this decision is based on the type of the breakpoint and the state of your program's variables.

Simple Breakpoints

These are one-off breakpoints which, when executed, are cleared and cause MonST2C to be entered.

Stop Breakpoints

These are breakpoints that cause program execution to stop after a particular instruction has been executed a certain number of times. In fact a simple breakpoint is really a stop breakpoint with a count of one.

Count Breakpoints

Merely counters; each time such a breakpoint is reached a counter associated with it is incremented, and the program will resume.

Permanent Breakpoints

These are similar to simple breakpoints except that they are never cleared - every time execution reaches a permanent breakpoint MonST2C will be entered.

Conditional Breakpoints

The most powerful type of breakpoint which allow program execution to stop at a particular address only if an arbitrarily complex set of conditions apply. Each conditional breakpoint has associated with it an expression (conforming to the rules already described). Every time the breakpoint is reached this expression is evaluated, and if it is non-zero (i.e. true) then the program will be stopped, otherwise it will resume.

Alt-B — Set Breakpoint

This is a window command allowing the setting or clearing of breakpoints at any time. The line entered should be one of the following forms, depending on the type of breakpoint required:

<address>

will set a simple breakpoint.

<address>,<expression>

will set a stop breakpoint at the given address, after it has executed <expression> times.

<address>,=

will set a count breakpoint. The initial value of the count will be zero.

<address>,*

will set a permanent breakpoint.

<address>,?<expression>

will set a conditional breakpoint, using the given expression.

<address>,-

will clear any breakpoint at the given address.

Breakpoints cannot be set on addresses which are odd or unreadable, or in ROM, though ROM breakpoints may be emulated using the Run Until command.

Every time a breakpoint is reached, regardless of whether the program is interrupted or resumed, the program state is remembered in the History buffer, described later.

Help — Show Help and Breakpoints

This displays the text, data and BSS segment addresses and lengths, together with every current breakpoint. Alt-commands are available within this display.

Ctrl-B — Set Breakpoint

This sets a simple breakpoint at the start address of the current window, so long as it is a disassembly window. If a breakpoint is already there then it will be cleared.

U — Go Until

This prompts for an address, at which a simple breakpoint will be placed then program execution resumed.

Ctrl-K — Kill Breakpoints

This clears all set breakpoints.

Ctrl-A — Set Breakpoint then Execute

A command that places a simple breakpoint at the instruction *after* that at the PC and resumes execution from the PC. This is particularly useful for DBF-type loops if you don't want to go through the loop, but just want to see the result after the loop is over.

Ctrl-D — GEMDOS Breakpoint

This allows a breakpoint to be set on specific GEMDOS calls. The required GEMDOS number should be entered, or a blank line entered if any existing GEMDOS breakpoint needs to be cleared.

History

MonST2C has a *history buffer* in which the machine status is remembered for later investigation.

The most common way of entering data into the history buffer is by using you single-step but, in addition, every breakpoint reached and every exception caused enters the machine state into the buffer. Various forms of the Run command also cause entries to be made into this buffer.

Note

The history buffer has room for five entries - when it fills up, the oldest entry is removed to make room for the newest entry.

H — Show History Buffer

This opens a large window displaying the contents of the history buffer. All register values are shown including the PC as well as a disassembly of the next instruction to be executed.