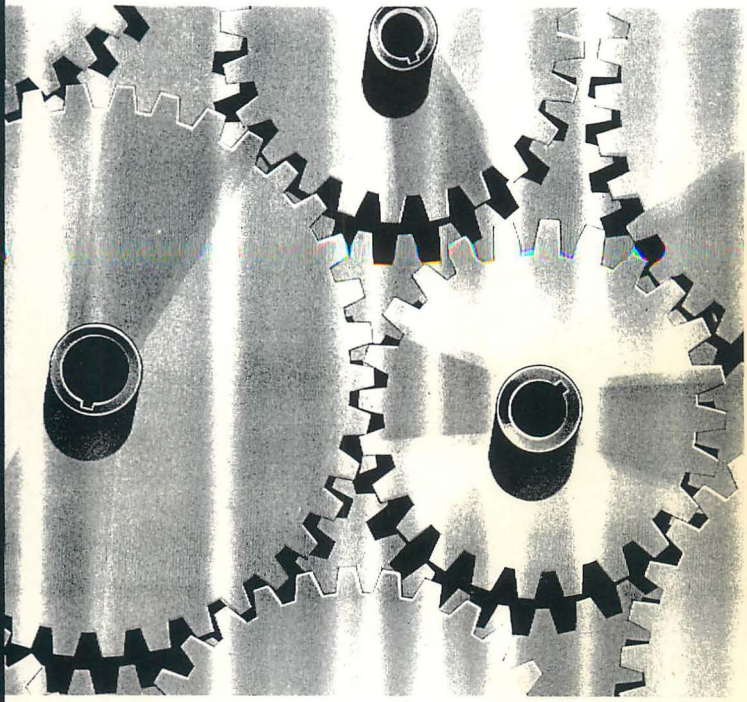
WERCS



MichTron®

WERCS

A Complete, Easy to use GEM Resource Construction Set

From HiSoft

Published by MichTron Inc.

576 South Telegraph
Pontiac, Michigan 48053

(313) 334-5700

BBS: (313) 332-5452

WERCS

A Complete, Easy to use GEM Resource Construction Set

For all Atari ST computers

Reference Manual

Software by HiSoft

Published by MichTron Inc.
576 South Telegraph
Pontiac, Michigan 48053

(313) 334-5700
BBS (313) 334-5452

ISBN 0-944500-25-0

YOUR RIGHTS AND OURS: *This copy of WERCS is licensed to you. You may sell your copy without notifying us. However, we retain copyright and other property rights in the program code and documentation. We ask that WERCS be used either by a single user on one or more computers or on a single computer by one or more users. If you expect several users of WERCS on several computers, contact us for quantity discounts and site-licensing agreements. Also if you intend to rent this program, or place this program on a BBS, contact us for the appropriate license and fee.*

We think this user policy is fair to both you and us; please abide by it. We will not tolerate use or distribution of all or part of WERCS or its documentation by any other means.

LIMITED WARRANTY: *In return for your understanding of our legal rights, we guarantee WERCS will reliably perform as detailed in this documentation, subject to limitations here described, for a period of thirty days. If WERCS fails to perform as specified, we will either correct the flaw(s) within 15 working days of notification or let you return WERCS to the retailer for a full refund of your purchase price. If your retailer does not cooperate, return WERCS to us. While we can't offer you more cash than we received for the program, we can give you this choice: 1) you may have a cash refund of the wholesale price, or 2) you may have a merchandise credit for the retail price, which you may apply toward buying any of our other software. Naturally, we insist that any copy returned for refund include proof of the date and price of purchase, the original program disk, all packaging and documentation, and be in salable condition.*

If the disk on which WERCS is distributed becomes defective within the warranty period, return it to us for a free replacement. After the warranty period, we will replace any defective program disk for $5.00.

We cannot be responsible for any damage to your equipment, reputation, profit-making ability or mental or physical condition caused by the use (or misuse) of our program.

We cannot guarantee that this program will work with hardware or software not generally available when this program was released, or with special or custom modifications of hardware or software, or with versions of accompanying or required hardware or software other than those specified in the documentation.

Under no circumstances will we be liable for an amount greater than your purchase price.

Please note: Some states do not allow limitations on how long an implied or express warranty lasts, or the exclusion or limitation of incidental or consequential damages, so some of the above limitations or exclusions may not apply to you.

UPGRADES AND REVISIONS: *If you return your information card, we will notify you if upgrades to WERCS become available.*

FEEDBACK: *Customer comments are VERY important to us. We think that the use, warranty and upgrade policies outlined above are among the fairest around. Please let us know how you feel about them.*

Many of the program and documentation modifications we make result from customer suggestions. Please tell us how you feel about WERCS - your ideas could make the next version better for all of us.

Table of Contents

1 Introduction

1.1 What is WERCS?

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 on the Atari ST. **WERCS** also produces header files for all the popular languages and comes complete with example programs to make programming with GEM resources much easier than it has been previously.

WERCS has many unique features not found in other resource editors and this manual contains a great deal of information that is otherwise not generally available.

1.2 Always make a backup

Before using **WERCS** you should make a backup copy of the distribution disk and put the original away in a safe place. It is not copy protected to allow easy backup and to avoid inconvenience. This disk may be backed up using the Desktop or any backup utility. The disk is single sided but may be used in double sided drives.

Before hiding away your master disk make a note in the box below of the serial number written on it. You will need to quote this if you require technical support.

Serial No:

1.3 Registration Card

Enclosed with this manual is a registration card which you should fill in and return to us after reading the license statement. Without it you will not be entitled to technical support or upgrades. Be sure to fill in all the details, especially the serial number and version number.

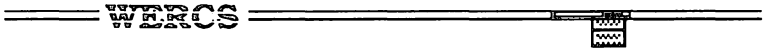
1.4 WERCS Disk Contents

The supplied single sided 3.5" disk contains these files:

WERCS.PRG	the resource editor
WERCS.RSC	the resource file
WERCS.LNG	used by **WERCS** to output header files
WERCS.INF	used by **WERCS** to keep track of your preferences
WIMAGE.PRG	a utility program for importing parts of Degas and
WIMAGE.RSC	Neochrome files
WCONVERT.PRG	A utility program to convert identifier files from the Kuma and Digital Research Resource Construction programs
WRSC.RSC	the main example resource file
WRSC.HRD	the names file for WRSC.HRD
WTEST.BAS WTEST.S WTEST.MOD WTEST.PAS WTEST.C	The example program in various languages.
README.TXT	See below for details
CHECKST.PRG	A utility that gives detailed information on the machine that you are using. See **Appendix 3** on technical support.

1.5 The README File

As with all HiSoft products **WERCS** is continually being improved and the latest details that cannot be included in this manual may be found in the README.TXT file on the disk. This file, if present, should be read at this point, by double clicking on its icon from the Desktop and then clicking on the Show button. You can print it by clicking on the Print button.



1.6 How to Use this Manual

This manual contains a great deal of information, much of it not generally available. However, fortunately, you do not need to read all of it to start using **WERCS** and using the resource files it generates in your own programs.

We realize that many people may have stopped reading this manual already; if you are about to stop reading and start experimenting with **WERCS** then please read at least **Section 3.3** because there is important information there on how item selection works *inter alia*.

We believe that to get the best out of **WERCS** you should follow the advice below on using this manual. Please read either the beginners section or the experts section; there isn't much point in reading both.

By *beginner* we mean someone who has not programmed using GEM resource files before; this includes C and assembler programmers with many years programming experience.

By *expert*, we mean someone who has extensively used another GEM Resource Construction Set and knows about GEM Resources, but may or may not have much programming experience.

1.6.1 Beginner's use of the manual

The rest of this chapter describes the basic theory behind the structure and use of resource files. Please read this next.

Chapter 2 is a Quick Tour of **WERCS** and is certainly the place to start to give you an overview.

After you have read **Chapter 2**, it is probably best to remake the example program we supply using your favorite language of the moment to ensure that you understand the mechanics of recompiling a program with a resource file. There is nothing worse than realizing that what you thought was a problem understanding GEM turns out to be a problem in your use of the compiler. We hope this program will provide a starting point for your own resource based programs.

Chapter 3 describes the use of **WERCS** in general including all the menu commands. If you don't like reading manuals please read at least **Sections 3.1** to **3.3** and **Section 3.11** this will save you a lot of time.

Chapter 4 describes exactly which objects have what attributes; you may want to refer to this as you get more adventurous with your use of resource files.

Chapter 5 describes the layout of resource files in memory; use this when you want to manipulate these within your programs. This chapter also contains details on the example programs, the format of the source files produced by **WERCS** and some hints on the more common errors when programming GEM and how to avoid them.

Chapter 6 describes the conversion utilities for the name files of other Resource Construction Sets and for importing Neochrome and Degas images.

Appendix 1 gives a summary of the **WERCS** keyboard shortcuts; these can also be found on the menus.

Appendix 2 gives information on the file formats used by **WERCS**.

Please read **Appendix 3** if you need to contact us for technical support.

And finally the index; if you are interested in looking up a range of topics you may find the **Table of Contents** at the front of the manual useful.

1.6.2 Expert's use of the manual

The rest of this chapter describes the basic theory behind the structure and use of resource files; you can probably skim through it.

Chapter 2 is a Quick Tour of **WERCS**, certainly the place to start for a swift overview of the way that the product works.

After you have read **Chapter 2**, you may want to convert the name files of some of your existing resource files next; this is detailed in **Section 6.1**.

It is probably best to remake one of your programs after loading and saving the resource file with **WERCS**. If you have problems then remake one of the example programs to ensure that you haven't made a mistake in the mechanics of recompiling.

A resource file contains such things to deliberately keep them separate from your program code. In addition, the X-Y coordinates 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 of individual *objects*, where each object has a distinct type, use, purpose and appearance.

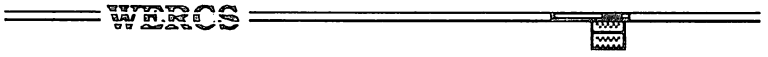
Whilst you are learning to use **WERCS** we recommend that you start off by using just Forms.

1.8 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:



Chapter 3 describes the use of **WERCS** in general, including all the menu commands. If you don't like reading manuals please read **Section 3.3** and **Section 3.11** - this will save you a lot of time. Some of the facilities you will not have seen in a resource file editor, particularly the Extras, Find String and Find Name commands.

Chapter 4 describes exactly which objects have what attributes; you may find some new and useful facts in here.

Chapter 5 describes the layout of resource files in memory, details on the source files produced by **WERCS** and the example programs supplied and also some hints on the more common errors encountered when programming GEM and how to avoid them. It also includes information on using programmer defined objects and "rolling-your-own" desktops.

Chapter 6 describes the conversion utilities for the name files of other Resource Construction Sets and for importing Neochrome and Degas images.

Appendix 1 gives a summary of the **WERCS** keyboard shortcuts; these can also be found on the menus.

Appendix 2 gives information on the file formats used by **WERCS**, including how to customize the header file output to use a language that **WERCS** doesn't already support.

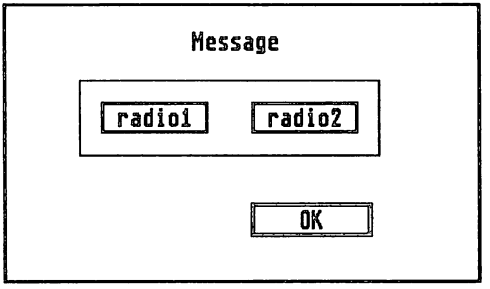
Please read **Appendix 3** if you need to contact us for technical support.

And finally is the index; if you are interested in looking up a range of topics you may find the **Table of Contents** useful.

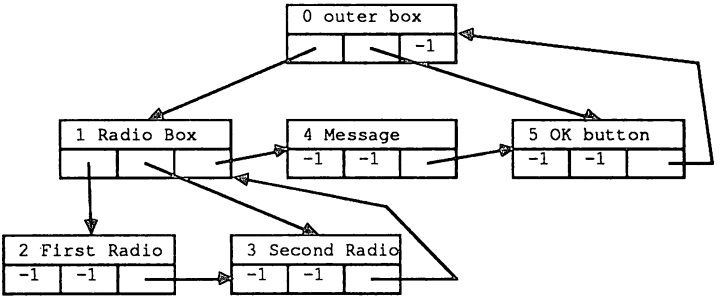
1.7 What is a Resource File?

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



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; it's children are First Radio and Second Radio; its *siblings* are Radio Box 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 **Section 5.1**.

1.9 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 you program (Exit). To start off with you need not be too concerned about flags and states as **WERCS** gives you sensible defaults. For more detail see **Section 4.2** and **Section 4.3**.

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: color, 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 colored 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 bitmapped 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 predefined objects and know a good deal about the low level interface to the programming language you are using.

1.10 Using a Resource File

1.10.1 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 exact format of the header file will vary, depending on which programming language you are using, but it is normally *included* in your program at compiletime.

If the compiled version of the header file is out of step with the resource file, strange things will happen; this varies from slight misbehavior to total system crashes. Details of the programming interface to a resource file can be found later, in **Section 5**.

2 Quick Tour

2.1 Running WERCS

The supplied master disk is rather full. If you only have one drive it is recommended that you delete some of the example files (the ones that are not relevant to your programming tools) before using **WERCS**. Remember, only delete them from your *backup* copy!

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 chapter.

2.1.1 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.

2.2 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.

2.2.1 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 evel 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.

2.2.2 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 resize it click on its lower right 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. The extension used for the header file will depend on the language you selected, detailed in **Section 5.4**.

3 Using WERCS

3.1 General

Most of the editing actions inside **WERCS** are obtained from menus or 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. There is a summary of the keyboard short-cuts in **Appendix 1**.

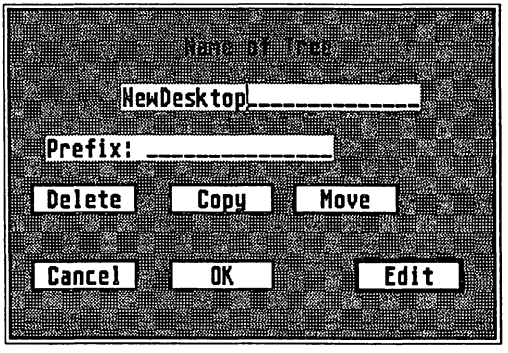
3.2 Introduction to Creating and Editing Trees

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: 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 will next be prompted with a dialog box as shown above, to enter the name for this 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 in **Section 3.11**.

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.

3.3 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.

3.3.1 Selecting objects

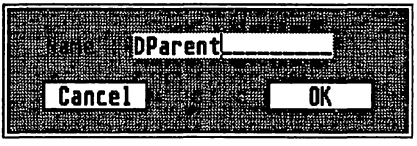
To change the attributes of any object, single-click on it. It will then be *selected* (highlighted) and you can use most of the menus to change its attributes, the border for example. The exceptions to this are the text items (see **Section 3.3.2**), Images (**Section 3.3.4**) and Icons (**Section 3.3.5**)- 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.

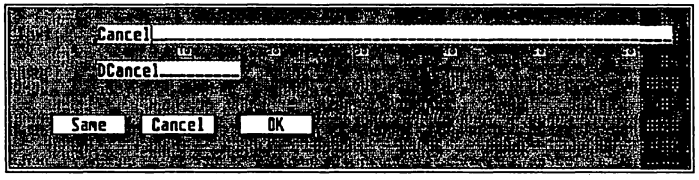
If you wish to edit the *parent* of the current object, single-click with the ALT key held down - the parent of the object will then be selected. If you already have an item selected and ALT click again, its parent will be selected. This may be repeated any number of times until the whole tree is selected.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

3.3.2 Item Names and Text

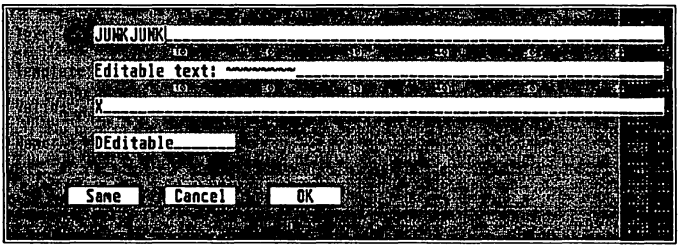
To change the Text or Name (remember: Text is the displayed message; Name is what your program will know the item as) of an object, double click on that item - this will present a Text dialog box. This varies depending on the item. For example a Box only has a Name and presents a dialog box like this:



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.

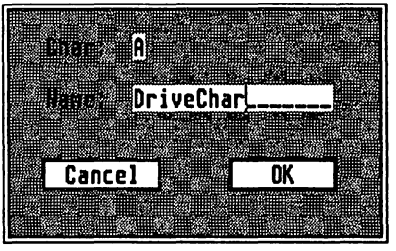
In the unlikely event that you need to enter two consecutive back-slashes type three instead; for three back-slashes type four and so on.

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 will 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, detailed in **Section 3.8**.

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



3.3.3 Moving and Sizing Objects

To change the size of an item, place the mouse near its bottom right 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 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 (Section 3.3.1)** 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. This will let you move a new copy of the object leaving the old one where it was. 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 move an object outside its parent then you will be asked for confirmation of this (unless you are in Expert Mode).

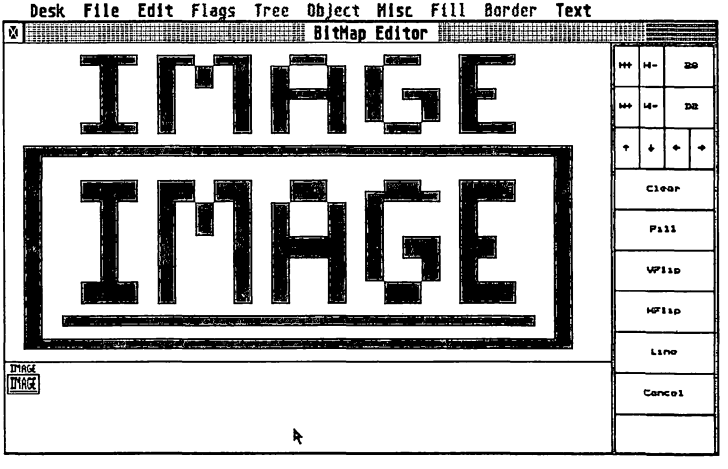
If you move an object so that it would completely cover another object or objects then you will again be asked if you wish to adopt these objects as children of the object that you have moved. If the new position of the object will partially cover another then you will be given an error message.

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. See **Section 3.5.2**.

3.3.4 Editing Images

Double clicking on Images will bring up the Icon Editor which will produce a screen display similar that shown 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 color 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.

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 reappear.

The maximum width of an Image that can be edited is 128 pixels.

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 (see **Section 3.10.10**) 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.

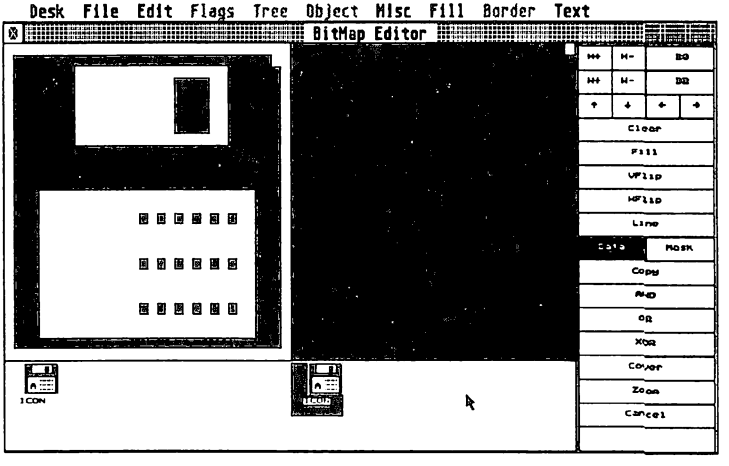
To name an Image, double click in the Actual Size area; the usual name box will then be displayed.

The Text menu can be used to set the foreground color 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.

3.3.5 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 bitmap 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 coordinates of the object. The object's coordinates are changed as usual in the Object Level window.

Frequently Icons do not need either or both of the String and Character; you can just set these to be blank. So that you 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.

3.4 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.

3.4.1 New

To starting editing a new empty file, click on the New item from the File menu. This is just like Quitting and reloading **WERCS**.

3.4.2 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 that supports GEM 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. The conversion utility WCONVERT is described in **Section 6.1**.

3.4.3 Importing Images

Images and Icons created using the WIMAGE utility (see **Section 6.2**) 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.

3.4.4 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 Modula-2, for example, **WERCS** will know each time you edit it. The .HRD file format is described in detail in **Appendix 2**.

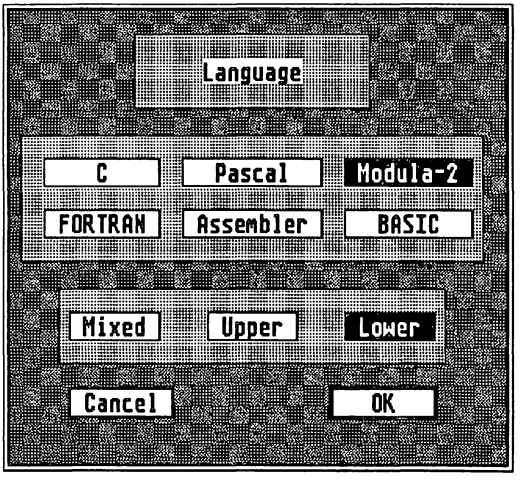
3.4.5 Save Prefs

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.

3.4.6 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 whether the names should be in lower, upper or mixed case. Details of supported languages and file extensions may be found in **Section 5.4**.

3.4.7 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.

3.5 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 checked. 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

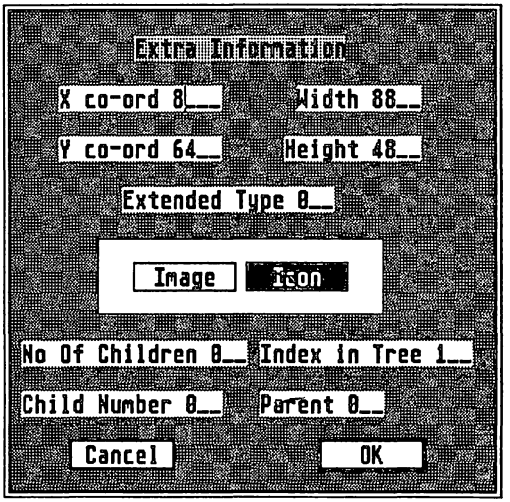
For details of the action of each flag, see **Section 4.2**.

3.5.1 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 select them once more.

3.5.2 Extras

This displays a dialog box similar to that shown below. This 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 'grandchildren'.

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 and make this child 0 then select the second and make this child 1, etc.

Objects may also be reordered using Sort from the Misc menu. See **Section 3.10.8** .

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.

3.6 Fill Menu

The Fill menu lets you change the fill pattern of the object, the color 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 color 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 colors of icons.

3.7 Border Menu

Lets you change the color 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 effects the size of the box that is drawn when the object is selected thus increasing or decreasing the visual size of the object.

3.8 Text Menu

The Text menu lets you change the justification, color, and size of the text of object types BoxChar, BoxText, FBoxText, Text and FText. The actual text is changed by double clicking on the object - see **Section 3.3.2.**

3.9 Clipboard

This 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.

3.9.1 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.

3.9.2 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.

3.9.3 Copy

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

3.9.4 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.

3.9.5 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.

3.9.6 Delete

Delete works like Cut (see **Section 3.9.1**) but leaves the contents of the Clipboard intact.

Hint:

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.

3.10 Misc Menu

3.10.1 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.

3.10.2 Auto Naming

If Auto Naming is enabled (shown by a check) 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.

3.10.3 Auto Snap

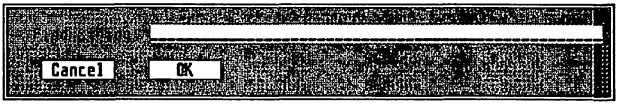
If this item is selected from the Misc menu then every item that you move or size will be *snapped* to the nearest character boundary. This is useful to make sure that items line up and will appear the same in different screen resolutions.

3.10.4 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.

3.10.5 Find Text

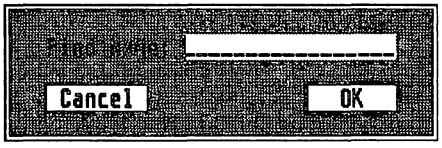
This enables you to find occurrences of a particular string within the text fields of the objects. You are presented with a dialog box, as below,



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

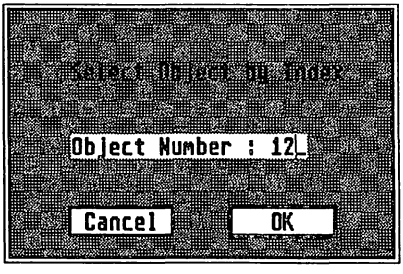
3.10.6 Find Name

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:



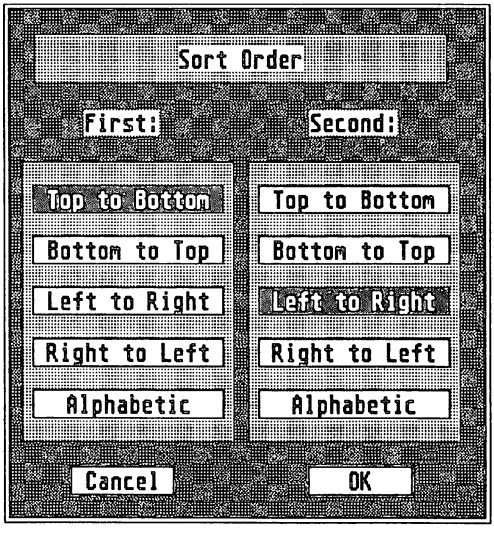
3.10.7 Number Select

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 of its parent.



3.10.8 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, objc, 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; see **Section 3.3.1**.

3.10.9 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. 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**.

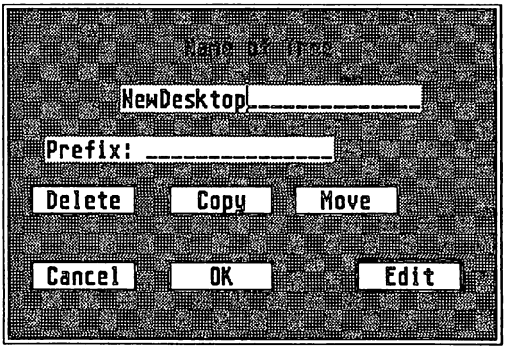
3.10.10 Expert level

If this is enabled (shown by a check) then all warnings to do with tree reorganizing 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.

3.11 Tree Level Editing

3.11.1 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. See **Section 3.3** for more details of Object Level editing.

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.

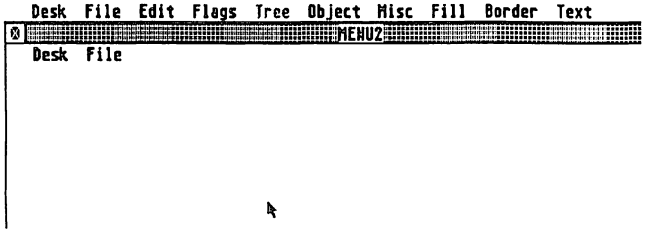
This box also lets you set up the Prefix for this particular tree. This is used to provide the start of the names of objects if you use Auto Naming.

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

3.11.2 Menus

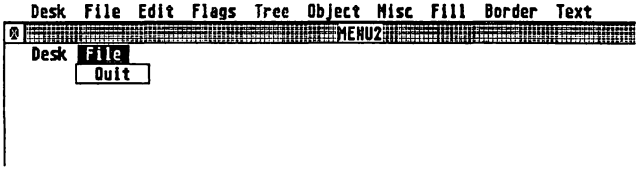
Menus are a very special type of Form which must conform to a number of un-published rules or GEM will behave strangely. Fortunately when using **WERCS** you don't have to worry about these rules as **WERCS** will cope with them for you.

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 where in the menu bar 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 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.

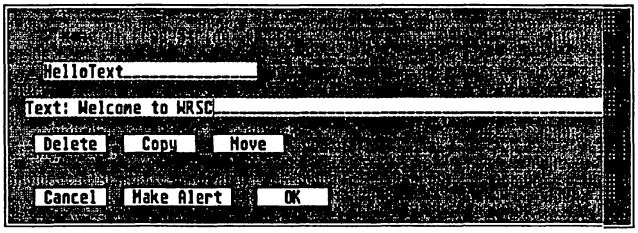
If you want your menu to work in more than one resolution don't use Icons or Images or you will find that there will either be gaps between them or they will overlap. This is because the width and height of these objects are a different number of character cells in different resolutions. This can be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

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

3.11.3 Free Strings

A Free String is a string of characters that is not connected with any particular tree. They can be used to make foreign-language versions of software very easy, 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. See **Section 3.3.2** for details.



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 in **Section 3.11.1**.

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.

3.11.4 Alerts

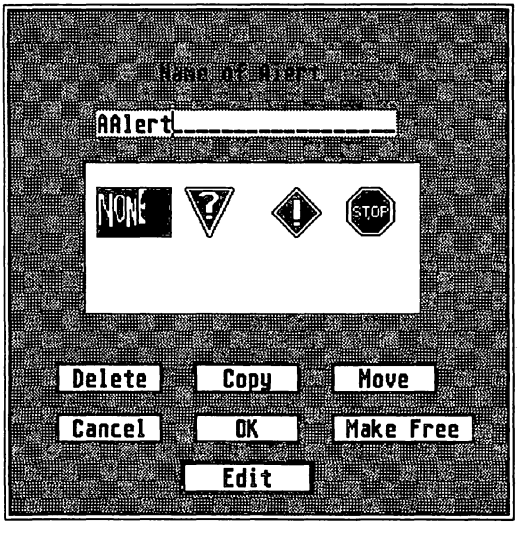
Alert Boxes are actually stored as Free Strings (see **Section 3.11.3** above) but are passed to the AES form_alert call to display an alert box.

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 1.02 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 1.00 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 1.00 ROM machine. Subsequent ROM releases (1.02 , a.k.a. *blitter ROM* and 1.04) 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 selected 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 when you are editing a Form.

You should only add Strings and Buttons to the Form and these will be automatically repositioned 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.

Reordering the Buttons in an Alert Box is achieved by dragging a Button. If you drag button A onto Button B then the Buttons will be rearranged 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

`linen` are the various message lines and `buttonn` 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.

3.11.5 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. See **Section 5.1.8** for the details of the BITBLK structure.

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.

4 Object Reference

This chapter details the different types of objects. For more detailed information regarding memory layouts of data structures, please see **Section 5.1**.

4.1 What is an Object?

There are thirteen different types of objects as follows:

4.1.1 Box

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

4.1.2 BoxChar

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

4.1.3 BoxText

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

4.1.4 Button

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

4.1.5 FBoxText

An FBoxText object is a formatted BoxText object; in addition to the normal Text attributes, it also has border attributes and a template and two extra strings for text entry.

These extra strings are called the Template and Valid fields. The AES displays the Template string as if it where displaying any other type of text except that for every underline character it displays a character from the main text string. For example, a date field object might consist of a Text entry of

```
011088
```

and a Template of

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

this would be displayed as

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

if it were an FBoxText.

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

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

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

In the above A-Z includes non English capital letters. 128-255 means that all characters greater with value greater than 128 can be used including lower case non-English letters and the £ sign. Note that the f validation character is not normally documented but is present in all versions of the operating system that we have used.

You can use different validation characters in the same string if you wish.

Thus for the date example above we would use the 9 character for all 6 character positions since the only characters allowed in dates are digits.

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

Also the pathname options (P and p) are of limited value as a number of software producers sadly use illegal pathname characters such as - in their filenames. All but the X, F and f validation characters also have the undesirable feature with the first, 1.00, operating system ROMs of crashing when you press _!

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

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

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

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

4.1.6 FText

Similar to a FBoxText (see **Section 4.1.5** above) but without border attributes.

4.1.7 IBox

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

4.1.8 Icon

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

4.1.9 Image

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

4.1.10 ProgDef

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

4.1.11 String

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

4.1.12 Text

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

4.1.13 Title

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

4.2 Flag Types

The different flag types for objects are as follows:

4.2.1 Selectable

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

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

4.2.2 Default

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

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

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

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

4.2.3 Exit

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

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

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

4.2.4 Editable

The Editable bit should only be set for the FText and FBoxText objects; this indicates that the user may edit the text in this field. For other types of objects the AES will misbehave often causing the system to bomb. The fields in the TEDINFO structure used by the AES for FText and FBoxText objects must conform to strict rules as described in **Section 4.1.5**.

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

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

4.2.4 Radio Button

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

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

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

4.2.5 Touch Exit

The Touch Exit bit is used to tell the AES to exit form_do when the user moves the mouse pointer over an item and clicks on it. The exit occurs when the mouse button is pressed down (rather than released in the case of the Exit flag). Touch Exit also differs from Exit in that the button need not be Selectable.

This flag may be used with any kind of object.

4.2.6 Hide

The Hide bit is used to hide an object and all its children from the AES. This means that the object is not displayed by objc_draw and will not be found by objc_find.

This is useful when you wish to remove part of a tree temporarily, without reorganizing that tree. For example, **WERCS** itself uses this facility when drawing the Extras dialog box to ensure that only appropriate types of objects are shown.

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

4.3 Flag States

The flag states for objects are as follows. For details of how they are stored in memory see **Section 5.1.3.**

4.3.1 Selected

If the Selected flag is set, it indicates that the object will be displayed highlighted.

This bit is changed from 0 to 1 or from 1 to 0 if the object is Selectable (see **Section 4.2.1**) when the user clicks on the appropriate object. Any type of object may have this bit set.

4.3.2 Crossed

The Crossed bit causes the AES to draw a white diagonal cross through the object. If the object is Selected then the cross is displayed as black.

This flag can be used on all objects except IBoxes.

4.3.3 Checked

If the checked flag is set the AES will draw the object with a black check mark, 4, inside it with the check in the top left corner. When the object is Selected the check is shown in Black.

The Checked flag may be used for any type of object including IBoxes.

4.3.4 Disabled

If the Disabled flag is flag is set for an object then it is shown grayed, that is, with less intense color than normal.

In addition, Disabled objects may not be Selected when using form_do or as part of an Menu even if they have the Selectable bit set. Note, though, that Disabled Editable fields may be edited!

4.3.5 Outlined

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

4.3.6 Shadowed

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

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

4.4 Object, Flags and States Summary

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

	Fill Pattern	Fill Color	Xparent/ Opaque	Border Color	Border Size	Text Color	Text Attr
Box	4	4		4	4		
BoxChar	4	4		4	4	4	
BoxText	4	4	4	4	4	4	4
Button							
FBoxText	4	4	4	4	4	4	4
FText			4		4	4	4
IBox				4	4	4	4
Icon		4				4	
Image						4	
ProgDef							
String							
Text			4		4	4	4
Title							

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

	Default	Exit	Editable	Crossed	Disabled	Shadowed
Box	+	+	Y	4	4	4
BoxChar	+	+	Y	4	4	4
BoxText	+	+	Y	4	4	4
Button	4	4	Y	4	4	4
FBoxText	+	+	+	4	4	4
FText	+	+	+	4	4	6
IBox	+	+	Y	6	+	4
Icon	+	+	Y	4	4	6
Image	+	+	Y	4	4	6
ProgDef	+	+	Y	4	4	6
String	+	+	Y	4	4	6
Text	+	+	Y	4	4	6
Title	+	+	Y	4	4	6

Key:

4 Changes appearance of object and the behavior of the AES

\+ Changes the behavior of the AES but not the appearance.

6 Has no effect

Y Causes the machine to crash with bombs.

5 Programming with Resources

This chapter details the various data structures and object types, together with common AES programming algorithms. In order to remain as language independent as possible, structures are shown with types *word* (16-bit) and *longword* (32-bit) entries and all AES calls are referred to using the standard C names. All character strings are C style, i.e. null terminated.

5.1 TreeStructure

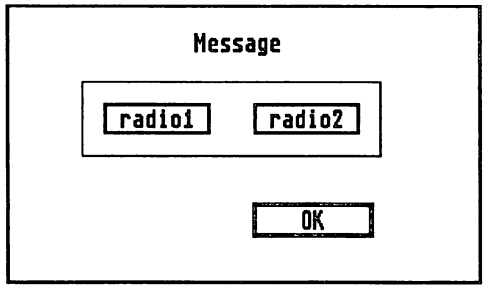
5.1.1 OBJECT Structure

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

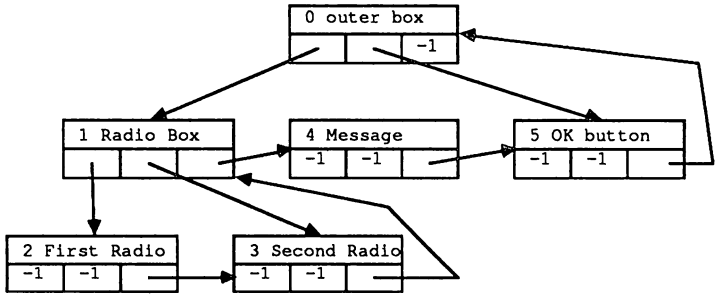
ob_next	word	index of object's next sibling
ob_head	word	index of first child or -1 if none
ob_tail	word	index of last child or -1 if none
ob_type	word	object type (high byte is ignore by the AES and used for extended object numbers)
ob_flags	word	
ob_state	word	
ob_spec	longword	depends on object type
ob_x	word	X coordinate of object relative to parent (in pixels)
ob_y	word	Y coordinate of object relative to parent (in pixels)
ob_width	word	width of the object in pixels
ob_height	word	height of the object in pixels

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

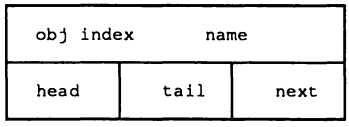
When it is loaded into memory an object tree is like an array of records. The first object (with index 0) is called the *root object*. It is normally the outer Box of a dialog box. Each object in the tree has three fields called ob_head, ob_tail and ob_next. These hold integer values that dictate to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



The tree structure this represents can be shown as:



where each box represents:



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

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

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

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

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

ob_head points to the first child (or is -1 if childless)
ob_tail points to the last child (or is -1 if childless)
ob_next points to the next sibling or if there are no more siblings to the object's parent.

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

Box G_BOX 20

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

Text G_TEXT 21

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

BoxText G_BOXTEXT 22

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

Image G_IMAGE 23

The ob_spec field contains a pointer to a BITBLT structure.

ProgDef G_PROGDEF 24

The ob_spec field contains a pointer to an APPLBLK structure.

IBox G_IBOX 25

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

Button G_BUTTON 26

The ob_spec field contains a pointer to the displayed string.

BoxChar G_BOXCHAR 27

The ob_spec field is used for the following fields:

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

String G_STRING 28

The ob_spec field contains a pointer to the displayed string.

FText G_FTEXT 29

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

FBoxText G_FBOXTEXT 30

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

Icon G_ICON 31

The ob_spec field contains a pointer to an ICONBLK structure.

Title G_TITLE 32

The ob_spec field contains a pointer to the displayed string.

5.1.2 Object Flags

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

Name on Menu	Standard Name	Bit	Mask
Selectable	SELECTABLE	0	1
Default	DEFAULT	1	2
Exit	EXIT	2	4
Editable	EDITABLE	3	8
Radio Button	RBUTTON	4	10
	LASTOB	5	20
Touch Exit	TOUCHEXIT	6	40
Hide	HIDETREE	7	80
	INDIRECT	8	100

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

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

5.1.3 Object States

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

Name on Menu	Standard Name	Bit	Mask
Selected	SELECTED	0	1
Crossed	CROSSED	1	2
Checked	CHECKED	2	4
Disabled	DISABLED	3	8
Outlined	OUTLINED	4	10
Shadowed	SHADOWED	5	20

5.1.4 Color Word

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

Border Colour		Text Colour		X/O	Fill Pattern		Fill Colour	
15	12	11	8	7	6	4	3	0

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

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

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

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

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

The L in the above names indicates *light*.

If you need to encode a color word into your program the best base to use is hexadecimal.

5.1.5 Border Thickness

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

5.1.6 TEDINFO Structure

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

te_ptext	longword	pointer to actual text
te_ptmplt	longword	pointer to template; editable portion denoted by underscores
te_pvalid	longword	pointer to string containing validation characters
te_font	word	font used: 3=system font, 5=small font
te_resvd1	word	reserved for future use
te_just	word	text justification required: 0=left, 1=right, 2=center
te_color	word	object color and pattern of box type objects (see previously for word format)
te_resvd2	word	reserved for future use
te_thickness	word	border thickness
te_txtlen	word	length of te_ptext string (*including* null)
te_tmplen	word	length of te_ptmplt string (*including* null)

5.1.7 ICONBLK Structure

This is used by the Icon object type only.

ib_pmask	longword	pointer to icon mask
ib_pdata	longword	pointer to icon data
ib_ptext	longword	pointer to the text displayed with the icon
ib_char	word	low byte is the displayed character, high byte defines color used. The top nibble is foreground color, bottom nibble is background
ib_xchar	word	X coordinate of ib_char relative to ib_xicon
ib_ychar	word	Y coordinate of ib_char relative to ib_yicon
ib_xicon	word	X coordinate of icon relative to the ob_x of the object
ib_yicon	word	Y coordinate of icon relative to the ob_y of the object
ib_wicon	word	width of the icon image in pixels (must be a multiple of 16)
ib_hicon	word	height of icon image in pixels
ib_xtext	word	X coordinate of icon's text relative to the ob_x of the object
ib_ytext	word	Y coordinate of icon's text relative to the ob_y of the object
ib_wtext	word	width of rectangle to display icon's text in (centered)
ib_htext	word	height of icon's text

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

5.1.8 BITBLK Structure

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

bi_pdata	longword	pointer to bit image
bi_wb	word even)	width of Image data in **bytes** (must be
bi_hl	word	height of Image in pixels
bi_x	word	source X coordinate
bi_y	word	source Y coordinate
bi_color	word	color word (see previously)

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

5.1.9 APPLBLK Structure

This is used by ProgDefs.

ab_code	longword	pointer to code to draw the object
ab_parm	longword	passed as a parameter to the low level drawing routine

5.1.10 PARMBLK Structure

This is passed to ProgDef drawing routines.

pb_tree	longword	pointer to start of object tree
pb_obj	word	the object index
pb_prevstate	word	the old state of the object to be changed
pb_currstate	word	the new (changed) state of the object
pb_x, pb_y	words	the pixel X and Y screen coordinates of the object
pb_w, pb_h	words	the pixel width and height of the object
pb_xc, pb_yc	words	the pixel X and Y screen coordinates of the current clip rectangle
pb_wc, pb_hc	words	the pixel width and height of the current clip rectangle
pb_parm	longword	copied from the address after the pointer to the drawing routine

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

5.2 Hints & Tips on Resources

5.2.1 Using ProgDefs

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

The drawing routine (planted in the ab_code field) will then be called whenever that object needs drawing or changing. The routine should normally be written in assembly language and conform to the following rules: Save all registers except D0; do not call the AES under any circumstances; call the VDI using the AES's VDI handle, found during initialization using graf_handle. On entry, the longword at address 4(A7) will be a pointer to a PARMBLK structure, detailed previously. On return, your custom drawing routine should place in register D0 the object state that you wish the AES to render over your object.If you

5.2.2 Creating New Desktops

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

A Form should be created in **WERCS** with the root object being a borderless Box with a suitable fill pattern and color. If any icons are required these should be added to this Form. The size of the Form is not relevant. To tell GEM to use this Form, the size and position (ob_x, ob_y, ob_width and ob_height) fields in the root object should be set to the usable screen size, found using the AES wind_get(0,WF_WORKXYWH, ... call. The form can then be installed using a wind_set(WF_NEWDESK, ... call with an object parameter of zero. Before your program terminates, the desktop must be deinstalled by passing an address of zero to the same call.

5.3 Common Mistakes and how to avoid them

The following is a list of common mistakes when programming with GEM and resources in general. The reasons given here are brief as there is insufficient space to expand upon them; they act as pointers for where to look in other documentation. Unfortunately we cannot provide support regarding these problems - it would be a full time job for a number of people!

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

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

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

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

Reason 2: If you do have editable text fields make sure that they conform to the rules in **Section 4.1.5** regarding editable text.

Problem: A GEM program (e.g. a compiler) crashes unexpectedly. After rebooting the same program works correctly under the same conditions.

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

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

Reason: Your graf_mouse calls are misbalanced in some way. For each hide (256) call you *must* have a show (257) call.

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

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

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

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

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

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

5.4 Language Details

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

5.4.1 Assembly Language

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

```
label EQU   1
```

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

```
    INCLUDE     TEST.I
```

The characters allowed in names are:

A-Z, a-z and _ as the first character and:

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

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

5.4.2 BASIC

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

```
CONST label%=1
```

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

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

The characters allowed in names are:

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

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

The .BH file is designed with **HiSoft BASIC** and **Power BASIC** in mind, and adaptation to other BASICs for the Atari ST is not straightforward. Most other BASICs for the ST are not suitable for serious GEM work in any case.

5.4.3 C

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

```
#define label 1
```

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

```
#include "TEST.I"
```

The characters allowed in names are:

A-Z, a-z and _ as the first character and:

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

All sixteen characters of the name are significant as is the case in the Manx **Aztec C**; compiler. This may cause problems with some C compilers if your names have the first eight characters the same.

5.4.4 FORTRAN

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

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

The characters allowed in names are:

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

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

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

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

5.4.5 Modula-2

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

```
DEFINTION MODULE TEST;

CONST label=1;

.....

END TEST.
```

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

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

```
IMPLEMENTATION MODULE TEST;

END TEST.
```

To access the constants from your other modules use:

```
FROM TEST IMPORT label;
```

or

```
IMPORT TEST;
```

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

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

The characters allowed in names are:

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

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

All sixteen characters of the name are significant.

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

5.4.6 Pascal

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

```
CONST

      label=1;

....
```

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

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

The characters allowed in names are:

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

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

All sixteen characters of the name are significant.

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

5.5 The WTEST Example Programs

5.5.1 Compiling WTEST

To illustrate the most common resource handling programming requirements, we supply with **WERCS** an example program written in a variety of languages. The programs all do the same thing but the implementations vary according to the language used.

If you are using a language implementation for which we do not supply an example and you wish to convert WTEST please read the rest of this section regarding the programming details. A ready to run version called WTEST.PRG is supplied on the master disk; it needs WRSC.RSC to run.

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

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

To find out how to actually recompile the program, read the section appropriate to the language that you are using below.

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

Assembly Language

WTEST.S is the source file for use with **DevpacST 2**. As well as the WRSC.I file produced by **WERCS** you will need GEMMACRO.S and AESLIB.S from your **DevpacST** master disk.

BASIC

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

C

WTEST.C is the **Aztec C** version; you will need the standard Aztec include files and you will need to link with the GEM library, g.lib.

Modula-2

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

Pascal

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

5.5.2 WTEST structure

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

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

Procedure INITIALIZE

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

Procedure SETDESK

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

Procedure DEINITIALIZE

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

Procedure HANDLE_DIALOG

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

Procedure SET_TEDINFO

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

Procedure GET_TEDINFO

Allows a particular TEDINFO structure to return its data portion.

Procedure SET_BUTTON

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

Procedure GET_BUTTON

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

Procedure TEST_DIALOG

This handles the particular dialog box in the resource file. It implements proper Cancelling - that is if it is cancelled then the Button state and Text entry are left alone. The observant amongst you will notice that this is not always implemented in our own products - it's a case of *do as we say, not necessarily as we do!*

Procedure HANDLE_MENU

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

Procedure MAIN

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

5.5.3 Implementation Specific Notes

These notes detail the differences in the different versions of the example program.

Assembly Language

WTEST.S is the **DevpacST 2** implementation. Although the code itself is easily convertible to other assemblers, it relies heavily on the AES macro and library files supplied with **DevpacST**. The executable version supplied is this version, because it is the smallest.

BASIC

WTEST.BAS is the **HiSoft BASIC** version which will also compile under **Power BASIC**. At the end of the program is a function newform_alert, required because the standard form_alert expects BASIC strings, not pointers to C strings.

C

WTEST.C is the **Aztec C** version which should be easily converted to other Cs assuming header and library files are available.

Modula-2

WTEST.MOD is the **FTL Modula-2** version. It uses the **FTL Modula-2** module Activity rather than evnt_mesag or evnt_multi. Modula-2 style names are used for the procedures.

Converting to other Modula-2s that use the standard C names for the AES functions and data structures should be straightforward although you will need to put in your own evnt_multi routine.

Pascal

WTEST.PAS is the **Personal Pascal** version. It differs quite considerably from the other versions due to the nature of the AES libraries supplied with the compiler. It uses variant records in many places to get around the type checking inherent in the Pascal language. Users of other Pascals may find converting the Modula-2 version an easier task than this version.

6 Utility Programs

6.1 WCONVERT Name File Converter

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

After its sign on message WCONVERT will present you with a file selector to enter the name file wish you wish to convert. If it has an extension of

DEF it is assumed to be a Digital Research RCS1 file

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

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

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

After **WCONVERT** has converted the file, you are given the opportunity to convert further files.

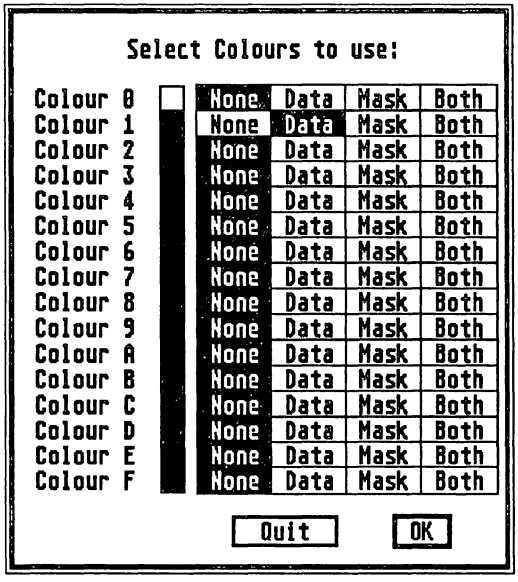
6.2 WIMAGE Image Convertor

WIMAGE is a utility for converting parts of Neochrome and DEGAS format files to resource files.

After double clicking on WIMAGE.PRG and after the sign on message you will be prompted to enter a file to convert via a File Selector. This file may be a NEOchrome format file (normally with an extension of .NEO) or uncompressed Degas/Degas Elite file (normally .PI3,.PI2 or .PI1). WIMAGE knows about medium and high resolution NEOchrome format files even though NEOChrome itself does not.

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

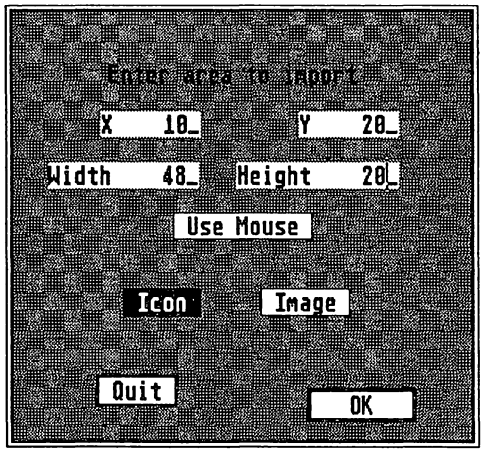
After entering the file name, the Image file will be loaded and you will be presented with a dialog box like this:



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

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

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



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

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

If you click on OK you will then be prompted to use a File Selector to enter the output file to which your Image or Icon will be written in the form of a Resource file. This can be imported into another resource file using the Import Image item from the **WERCS** File menu. See **Section 3.4.3**.

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

A 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		Center	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		
Undo	Cancel		
Backspace	Delete		

To help you remember these: 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. P is short for Programming Language, O for Order (Sort) ^G for Grayed (Disabled) and ^B for Border (Shadowed).

B File formats

B.1 HRD file format

.HRD files consist of a header record, any number of variable length data records and then an end-of-file record as sshown on the following pages.

B.1.1 HRD Header record

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

B.1.2 HRD Data Record

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

B.2 LNG file format

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

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

```
*LANGUAGE Modula-2
```

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

The end record is a single line:

```
*END
```

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

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

```
*SOURCE .MOD
```

This specifies the extension of the source file that **WERCS** will generate (`.MOD` in this example); the full stop (.) must be present.

```
*SIGNIFICANCE 16
```

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

```
*INITIAL   a-z,A-Z,_,$
```

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

```
*INITIAL a-zA-Z_$
```

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

```
*FOLLOW a-Z,A-Z,0-9,_,$
```

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

```
*INIT
*TREE
*OBJECT
*EXIT
```

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

`%F` base file name (without drive, directory or extension),

`%T` name of the current tree,

`%N` name of the current object,

`%V` the value of the current object (for `*OBJECT`) or tree (for `*TREE`),

`%%` a single % character.

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

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

C Technical Support

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

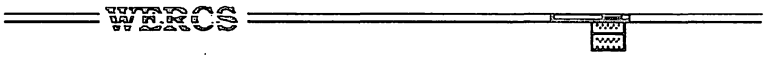
- **The Best Way** We are available in the MichTron RT on GEnie. Refer to the last page of this manual for more information.
- **Phone** Please try to call between 3pm and 5pm (Eastern Time), though calls will be accepted at other times. Please have your serial number handy. Telephone support is available **only to registered users.**
- **Mail** Send a disk with your program on it. Please put your name, address, and serial number on it.

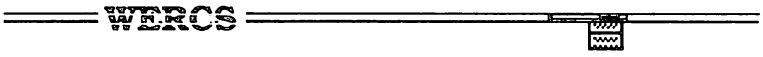
Whichever method you use please *always quote your serial number* (from your master disk) and the version number of the program. In the case of **WERCS** itself, this is given by the About WERCS menu item. We reserve the right to refuse technical support if you do not supply this information.

For bug reports, please run the `CHECKST.PRG` program supplied and quote the information given by it, as well as details of any desk accessories and auto-folder programs in use. If you think you have found a bug, try and create a small program that reproduces the problem. It is always easier for us to answer your questions if you send us a letter and, if the problem is with a particular source file, enclose a copy on disk (which we will return).

C.1 Suggestions

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





Index

Come and join us at the Roundtable, Where the GEnie and the Griffin meet!

Does this sound like a fantasy? Well, it may just be a dream come true! When General Electric's high-tech communications network meets MICHTRON's programmers and support crew, ST users around the country will hear more, know more, and save more.

We know that our low prices and superior quality wouldn't mean as much to you without proper support and service to back them up.

So we are available on GEnie, the General Electric Network for Information Exchange. GEnie is a computer communications system which lets you use your personal computer, modem, and communication software to gain access to the latest news, product information, electronic mail, games, and MICHTRON's *own* Roundtable (See the specialMicroDeal Section for game information)!!

The Roundtable Special Interest Groups (SIG) gives you a means of conveniently obtaining news about our current products, new releases, and future plans. Messages directly from the authors give you valuable technical support of our products, and the chance to ask questions (usually answered within a single business day).

GEnie differs from other computer communication networks in its incredibly low fees. With GEnie, you don't pay any hidden charges or minimum fees. You pay only for the time you're actually on-line with the MichTron product support Roundtable, and the low first-time registration fee.

For more information on GEnie, follow this simple procedure for a free trial run. Then if you like, have ready your VISA, Mastercard or checking account number and you can set up your personal account immediately -- right on-line!

1. Set your modem for half duplex (local echo)--300 or 1200 baud.
2. Dial **1-800-638-8369**. When connected, type **HHH** and press **Return**.
3. At the U#= prompt, type **XJM11957,GENIE** and press **Return**.

And don't forget, MICHTRON's Bulletin Board System, The Griffin BBS, is still going strong (the griffin is the half-lion/half-eagle creature on our logo). Our system is located at MICHTRON headquarters in Pontiac, Michigan. For a trial run, call (313) 332-5452.


MichTron®
576 S. Telegraph, Pontiac, MI 48053
Orders and Information (313) 334-5700