

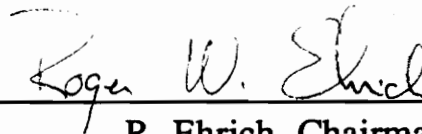
172
103
PC-Gipsy: A Usable PC-based Image Processing System

by

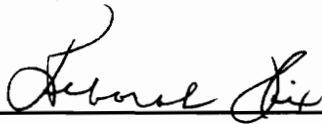
Karl Henry Melder

Project submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Information Science
in
Computer Science

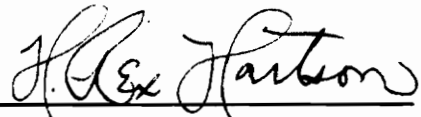
APPROVED



R. Ehrich, Chairman



D. Hix



R. Hartson

December 17, 1990
Blacksburg, Virginia

c.2

LD

5655

V851

1990

M 453

c.2

PC-Gipsy: A Usable PC-based Image Processing System

by

Karl Henry Melder

R.Ehrich, Chairman

Computer Science

(ABSTRACT)

The objective of this master's project is to improve the quality of the interaction that occurs between the image processing user and the Gipsy image processing system. Gipsy, in its present configuration, has some functional and user-interface deficiencies which affect usability. Changes in both the channel by which the user accesses Gipsy and improvements in the user interface may improve the overall usability of the system. These changes include the implementation of an IBM PC-based direct manipulation interface to Gipsy, which will enable the user to view and manipulate images and to build new Gipsy commands.

Acknowledgements

I would like to thank Dr. Roger Ehrich, my advisor and mentor, for his patience and support throughout my career at Virginia Tech. I would also like to thank Dr. Debby Hix and Dr. Rex Hartson for their enthusiastic support. Thanks to you I have found a career which ideally matches my personality and goals in life.

As far as the friends I made here at Tech - you guys are the greatest. I would like to thank TW, Matt, Jo, Horace, Rob, Mike, Ahmed, Birgit, Vincent, Hugues, Ajit, Christian, Rani, and Jim. A guy like me could not have better friends. Now I can't forget Robert. I cannot begin to express what our continued friendship has meant to me. Sorry about your address book!

And finally, my family. Mom, Dad, and Annette. I don't say it enough, but I love you with all my heart. I would not have made it without you.

Table of Contents

Abstract

1	Introduction	1
1.1	Objective	1
1.2	Approach	1
1.3	Expected Results	2
2	Description of Gipsy	4
2.1	What is Gipsy?	4
2.2	The Usability of Gipsy	6
3	Image Processing Users	11
4	PC-Gipsy	14
4.1	Why a Direct Manipulation Interface	14
4.2	Description of PC-Gipsy	15
4.3	User Customized Pulldown Menus	17
4.4	Scope of this project	19
5	Tools	21
5.1	DOS Extenders	21
5.3	Graphical User Interface Libraries	22

6	Development Issues for PC-Gipsy	27
6.1	IBM PC/AT Platform	27
6.2	Window Design Issues	28
6.3	Standard Image File (SIF) structure	33
7	Final Product	36
7.1	The Main Window	36
7.2	The Secondary Image Windows	38
7.3	The Window Manager	40
8	Conclusion	42
8.1	Design Conclusions	42
8.2	Future Work	43
	Bibliography	44
	Appendix	46
	List of graphical user interface libraries/toolkits for the PC ..	47
	The PC-Gispy Source Code	49
	Source Code for Spawning Tasks	168

1 Introduction

1.1 Objective

The objective of this master's project is to improve the quality of the interaction that occurs between the image processing user and the Gipsy image processing system. Gipsy is a general purpose image processing system which now resides on DEC VAX/VMS computers. It is designed to be highly adaptable and to have appeal to an audience from a variety of disciplines. It is, however, a system that has a high learning curve resulting from its generality and complexity. In addition, Gipsy's extensive and complex set of commands is easily forgotten, even after a period of only several weeks. In its present configuration it is also functionally inaccessible for those who are accessing Gipsy via a remote terminal.

1.2 Approach

Improvements in the human-computer interface could dramatically improve the overall usability of this system without requiring total redesign of Gipsy. Many of the commands which now require a commitment either to short-term or to long-term memory by the user could be made more intuitive, thereby improving usability both for the novice and for the intermittent user.

The interface could be customized to meet the specific needs of a predefined population of users. The user could customize a given interface to meet individual needs and expectations. Also, the ability to view images at the remote station (i.e., a microcomputer) would make the system available to a wider audience.

These improvements will be implemented using a graphical user interface (GUI) that utilizes a direct manipulation environment, such that a user initiates user-machine dialogue via a pointing device (e.g., mouse). There is anecdotal and subjective evidence to suggest that a direct manipulation environment provides an improved medium for human-computer interaction. Some empirical work confirms this belief and goes on to provide evidence for improved task completion times, decreased number of user errors, lower user fatigue, and a higher learning curve [Microsoft, 1990].

1.3 Expected Results

The expected product for this master's project is a first prototype of what will be an IBM PC-based Gipsy system that enables a user to view and manipulate images generated by Gipsy. This system will be able to display multiple windows, with each window containing an image that has been generated by Gipsy. The user can scroll through an image that may be much larger than the window itself and can manually resize and move those windows so that the user can compare two or more juxtaposed images.

By virtue of being implemented in a direct manipulation environment, PC-Gispy should improve the overall usability of the Gipsy system. In addition, the facility of being able to view and manipulate images on the PC will improve user access the the image processing system. The ability for the user to customize the

interface will also allow a user to optimize his own user-computer dialogue.

2 Description of Gipsy

2.1 What is Gipsy?

Gipsy is a software system that was developed in the Spatial Data Analysis Laboratory of Virginia Polytechnic Institute and State University for the purpose of meeting the needs of image processing users from a wide range of disciplines. The Gipsy system is capable of:

- using single or multiband images from a variety of sources (e.g., remote sensing devices, scanners),
- processing those images via a command line or menu driven interface,
- keeping a record of all the processing performed on any given image,
- providing the user with on-line documentation available from the Gipsy command-line,
- giving the user the capabilities to develop and add new commands, and
- providing full access to code for the existing commands.

The heart of Gipsy is the kernel, the section of software that handles all the machine-dependent activities such as memory allocation, process control, file manipulation, interrupts, and error

handling. The kernel creates a layer of abstraction above the operating system to enable the image processing software to be written in hardware-independent terms. This structure enhances the portability of Gipsy from one platform to another by limiting the amount of software that directly interacts with the operating system. In addition, Gipsy is written in a hardware-independent language called RATFOR [Kernigan and Plauder, 1976], for which there is a preprocessor that generates FORTRAN IV code. Gipsy currently resides on VAX computers operating VMS.

The user interface allows the user to issue commands either by a command-line or by a menu system. For the command-line driven interface, the user can issue a command that might have the following format:

```
gipsy> input > command > output (cflags, sflags)
```

The arrows indicate the flow of data. The output can specify filenames or display devices, and the input can be from input devices or files. A set of option flags can be included, which are either command-dependent (i.e., cflags) or command-independent (i.e., sflags). Any given command can initiate a series of queries which supply additional input arguments for that particular command. Each query has established default argument values. The user also has the option of creating macros (runfiles) in which a sequence of commands are assembled and input/output parameters are specified.

The menu driven interface provides a hierarchical organization to the Gipsy command library. Each panel (i.e., node) can have up to 13 commands to choose from. The menu system in Gipsy has the additional feature of providing a one line description of each command directly in the menu.

Presently, Gipsy has an extensible library of over 370 commands for performing various image processing tasks [Garland and Ehrich, 1987]. The user has the option of creating new commands and integrating them into the Gipsy command line facility. It must be emphasized that this process is encouraged, which thereby adds to the total body of knowledge available within the Gipsy system.

2.2 The Usability of Gipsy

Before any detailed discussion of Gipsy's usability, it would be useful to comment on the term usability. It is a relative term that remains rather ill-defined in the field of human-computer interface design. For some, a usable system is simply a functionally complete system. For others, a usable system is one that does not require a manual to learn or to operate. The author has attempted to provide a set of general goals for developing usable software. These include:

- 1) gives the user an effective means to navigate the available functions,
- 2) is designed to accommodate novice, intermittent, and expert users [Shneiderman, 1987],
- 3) allows the user to perform tasks efficiently,
- 4) is intuitive to the user, and
- 5) is functionally appropriate for the user.

This set of goals defines usability in terms of both the functional capabilities of the system and the user interface. A functionally appropriate system will not increase usability if the interface does not allow the user to access the functionality efficiently, and a well designed interface will not prove useful if the necessary functions to complete a given task are not available.

At the functional level, Gipsy has several weaknesses. First, although Gipsy resides on a multiuser/multitasking system and thus gives the user the convenience of remote terminal access, it does not allow the user to review the images it produces over such a remote terminal. An effective interaction can occur only if the user is within some laboratory facility where display devices are available.

Second, it is very difficult for users to integrate new commands and functions into the Gipsy command set [Ehrich, 1990]. For a user to write a new command involves strict guidelines and conventions for the documentation and code. The process requires the user to understand the general structure of the Gipsy kernel and the operating system in which the system resides. Programming a Gipsy command involves understanding the general command structure, file primitives, memory management, Input/Output, programming conventions, graphic primitives, and RATFOR.

Third, Gipsy's weaknesses become exacerbated by the program's large and complex array of commands. These three usability problems are further compounded by Gipsy's incomplete and context insensitive documentation system.

When Gipsy is in command-line mode, it has simply a flat list of commands with no conceptual guide for choosing commands. Many of the commands have been grouped under general headings, but the level of complexity and command terminology remains an issue for both the novice and the intermittent user. The advantages and disadvantages of command-line systems are described by Paap and Roske-Hoftstrand (1988):

- 1) Command-line systems force the user to learn and recall commands.
- 2) They do not prevent the user from improperly using an option out of context.

- 3) Command-line systems are generally highly flexible, permitting the user to reorder sequences of commands.
- 4) A command-line requires little screen space. The remaining screen space allows several previous commands to remain in view.
- 5) Commands-line systems are fast, but require more a priori knowledge than other types of user interaction systems.

The ability to use Gipsy successfully when in the command-line mode is dependent upon having a good understanding of the whole system, having immediate access to documentation, and having the ability to recall command names with their argument list.

The menu system, included in Gipsy to help the novice, does provide a means for the user to "discover" commands. Paap and Roske-Hoftstrand (1988) describe the advantages and disadvantages of menu systems:

- 1) A user need recognize only the command, and not the context or its parameters.
- 2) Menus guide the user in a step-by-step process and isolate the user from incorrect choices.
- 3) Menus have limited flexibility. A menu designed for one set of user tasks may be inappropriate for other user tasks.
- 4) Displaying any menu panel requires significant screen space.
- 5) A menu system requires less a priori knowledge but is slower to use.

Unfortunately, given the number of commands currently available in Gipsy, the size of a complete menu system could become unmanageable for the user. It would require a minimum of 74 menu panels assuming each panel is restricted to having five commands. This does not include the additional number of commands required to provide the user with a means to navigate the system, which will be determined by the structure imposed on the menu. For example, let us assume a two-level hierarchical structure is imposed. The first level contains all the commands which point to panels in the second level. The panels in the second level contain only Gipsy commands. For this configuration an additional 15 panels are required for a total of 89.

As a secondary issue we have the classic depth-breadth tradeoff which will affect the user's navigation of the menu hierarchy. The deeper the hierarchy, the greater the possibility of the user getting lost. To counteract this effect, the designer may choose to increase the menu's breadth. However, as breadth is increased, so is the amount of time the user needs to choose an item on the menu [Lee and MacGregor, 1985]. Increasing the menu's breadth is the approach used in Gipsy, where any given panel has up to 13 commands to choose from. The menu system in Gipsy provides the additional feature of having a one line description of each command. This affords the user an excellent means to locate suitable commands, but it may increase user search time. This menu format is highly suitable for those users being first introduced to the system but will quickly become burdensome as the user becomes more sophisticated.

The result of using a command-line interface is a system with a very high learning curve before a user can be considered an expert. The use of the menu system may improve a novice user's ability to navigate the system, but further studies are required before assuming a reduction in the learning curve has occurred.

As part of this research project, a study was performed using a population of several image processing users to identify common problems

users had with image processing systems [Melder, 1990]. Many of these users had some level of experience with Gipsy. All reported a high learning curve and difficulty in remembering the names of functions as primary drawbacks to the system. Those individuals who considered themselves experts at using Gipsy felt the high degree of flexibility the system offered compensated for any difficulties they had and that with better documentation the system would overcome these drawbacks. It is useful to note that these same users also felt the concept of ease-of-use and flexibility were inverse ideas, such that an increase in flexibility results in a decrease in ease-of-use.

3 Image Processing Users

Image processing is an interesting problem domain for studying system usability and generalizability in the context of human-computer interaction. The range of expertise in this field among users is broad, from the operator who uses turn-key image processing systems to monitor a production system, to a research scientist who creates new techniques for analyzing images. The range of expertise also varies over time for a given user, starting with a user approaching a new problem where the goals, tasks, and techniques are often "fuzzy" and ill-defined, to the point in development where the problem is defined and a solution is found. The goals and tasks can also vary from individual to individual, depending upon the application domain, the experience and knowledge of the user, the environment, and the system functionality.

This translates to a very complex environment for developing an appropriate and usable interface. Successful design and implementation therefore require a somewhat structured and documented approach where global issues, task dependent issues, and human frailties and strengths must be examined in detail.

As stated earlier, the range of users in this population is broad, the expertise varied, and the environment complex. The author interviewed several of these individuals for a previous study with the purpose of exploring the needs and goals of image processing users [Melder, 1990]. It became readily apparent, even with the relatively small sample of

individuals who were interviewed by the author, that these individuals had different interests, different levels of skill in image processing, and various perspectives on what type of systems would prove useful.

The results of the oral interview indicate that all the interviewees placed a high value on having a flexible imaging system that has good documentation. Interestingly enough, most were not critical of the complexity that was characteristic of the systems they used. Many had the opinion that they could work with any system that was presented to them. They assumed that complex and adaptable systems must have a high learning curve, and they assumed that such a system would be difficult to use. However, there was general agreement that a decreased learning curve would make the system more usable.

From the study performed with image processing users, these conclusions were derived to guide the design of an image processing system:

- Editing tasks and tasks to retrieve image information (e.g., pixel size) apply to a broad spectrum of users.
- An image processing system must be flexible despite its inherently high level of complexity.
- Emphasis must be placed on minimizing the learning curve and compensating for the intermittent user.
- An image processing system must be well-documented. It cannot rely on the user's intuition given the typically complex nature of image processing systems.
- The population of image processing users come from a wide range of disciplines with wide ranging levels of expertise.

- Terms, and their relation to other terms, used in image processing are often context- and domain-dependent.
- A user should be able to view changes in an image as a given function is applied, be able to view and compare multiple images, and be able to interact with the system using visual cues.
- Image processing systems should allow users to impose their own mental model for image processing.

The results from the study provides only the first step in what should be an iterative design approach. There should be a clear and continued focus on the end-user. This should involve extensive prototyping and usability studies throughout the software development cycle.

4 PC-Gipsy

The long-term goal is to produce an IBM PC-based version of Gipsy that is layered in a generic direct manipulation interface. This chapter will discuss the rationale for choosing a direct manipulation interface. It will also describe PC-Gipsy in its mature form, and PC-Gipsy as defined by this project.

4.1 Why a Direct Manipulation Interface?

As discussed earlier, the issue of menu versus command-line driven systems is mute in face of the complexity of the Gipsy system. The menu ultimately does not satisfy the novice, and the learning curve for the command driven system remains unbearably high. The direct manipulation type interface, however, may resolve this issue by providing an expert user with direct access to Gipsy commands while providing context dependent access to those commands.

The term direct manipulation was introduced by Ben Shneiderman (1982) as a means of describing user interfaces that have the following attributes:

- Continuous representation of a set of objects of immediate interest

- User performs physical actions with the system providing immediate feedback
- Reversible operations

A direct manipulation interface has been shown to be a highly efficient means for expert users to interact with a system [Zieger, 1988][Microsoft, 1989]. It also an efficient means for novice users to learn quickly the basic functionality and structure of a system as well as reducing anxiety and error rates for the user [Ziegler, 1988][Microsoft, 1989].

4.2 Description of PC-Gispy

Our goal, therefore, is to produce a PC-based Gipsy system supported by a direct manipulation type interface. PC-Gipsy will be structured such that the Gipsy kernel and the graphical user interface are integrated. This integration would allow the user to have full access to the Gipsy command set via the interface and enable the user to build and execute applications (Figure 1).

To execute an application, a message containing the command name, command options, and command arguments is prepared by the Gispy kernel and transmitted to the application as it is spawned. The kernel releases control of the display screen and passes execution control to the spawned application. The application may display graphic results until its execution halts, at which time both execution control and screen control return to the kernel.

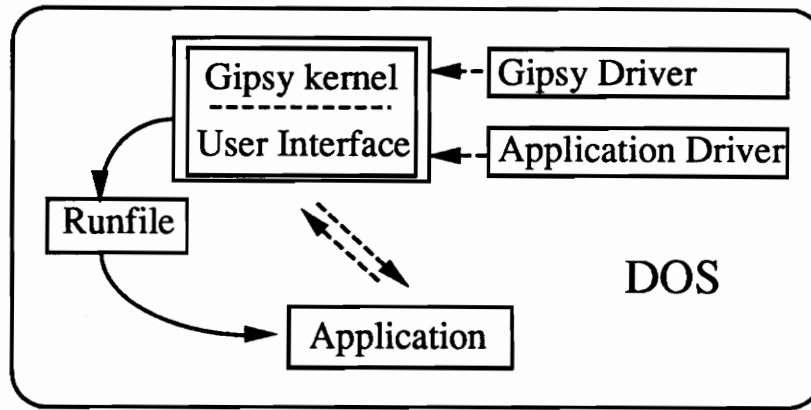


Figure 1: The structure of the PC-Gipsy system.

The application driver would contain the help facility, and the whole system would be layered in a DOS environment. The system would have the following elements:

- 1) The user interface would be a direct manipulation interface with the basic objects associated with a graphical user interface (GUI). These objects would include pulldown menus, dialog boxes, buttons, scrollable regions, and windows. The interface would be a multi-windowing system that could display several images concurrently. Each window could be resized or moved to any position on the screen. Each window would allow the user to scroll the image in either the vertical or horizontal plane.
- 2) The interface could be easily customized (packaged) to suit the generic needs of a predefined population of image processing users. The pulldown could be modified by the developer to reflect both the functional needs of the population as well as the terms this population might use.
- 3) The interface could be easily customized by the user to reflect individual needs and tasks. For example, the user would have the ability to define custom menus and interaction styles.

4) An on-line documentation management system would be available that would allow the end-user to change the documentation to better serve his needs.

In addition to the elements of the system described above, PC-Gipsy should also have the following additional functionality:

- 1) Real-time visualizations of data as an image transformation occurs [Ehrich, 1990]
- 2) Hackfiles to allow the user to define default settings for the working environment.
- 3) The ability to import and export images of different types.
- 4) The ability to spawn tasks.

4.3 User Customized Pulldown Menus

A pulldown menu that can be customized by the user is a useful feature that is already incorporated in major software products like Generic CADD. Researchers using a large system like Gipsy tend to use a small repertoire of commands in any given application, although there is usually a small core of commands common to each application. Among the commands supporting an application, several usually have been written by the end-user. By allowing the end-user to customize menus, a highly usable interface is often produced because it is small and because the user is the one who created it to begin with [Ehrich, 1990].

A custom command can be built in one of two ways. First, a user could build a runfile which would contain a sequence of Gipsy commands. The language for building a runfile would be similar to that used in the

current version of Gipsy, but with the added capability to invoke some low level user interface functions [Figure 2] to acquire additional information. The second way to build a custom command would be to encode it directly in C, using standard Gipsy kernel or library functions to integrate it into the Gipsy environment.

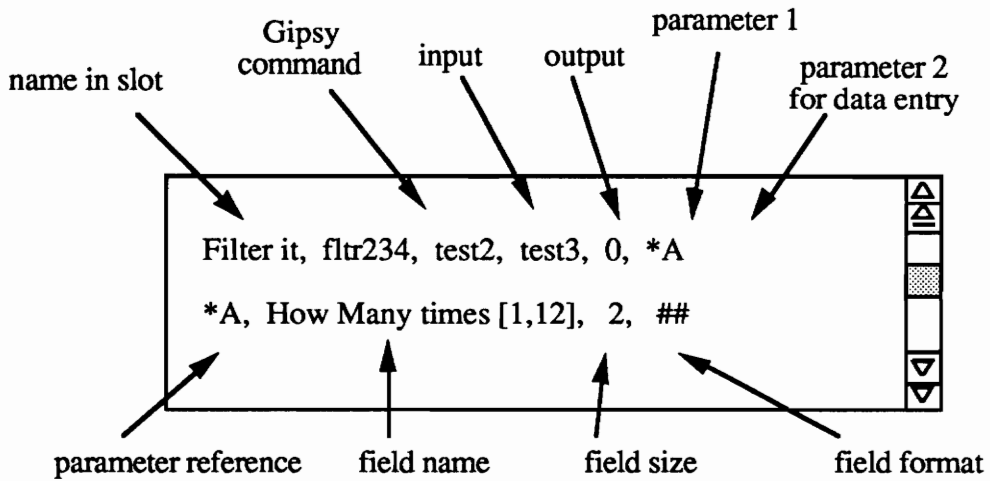


Figure 2: A possible syntax for a user customizable menu system.

The user would have access to the customized commands via the pulldown menu system. The user could simply edit a menu definition file to change the menu structure or use an on-line direct manipulation menu edit facility [Figure 3]. When an application is initiated, the parameters which indicate data entry requests become user dialogue boxes. This will allow the user to change input parameters via a direct manipulation interface.

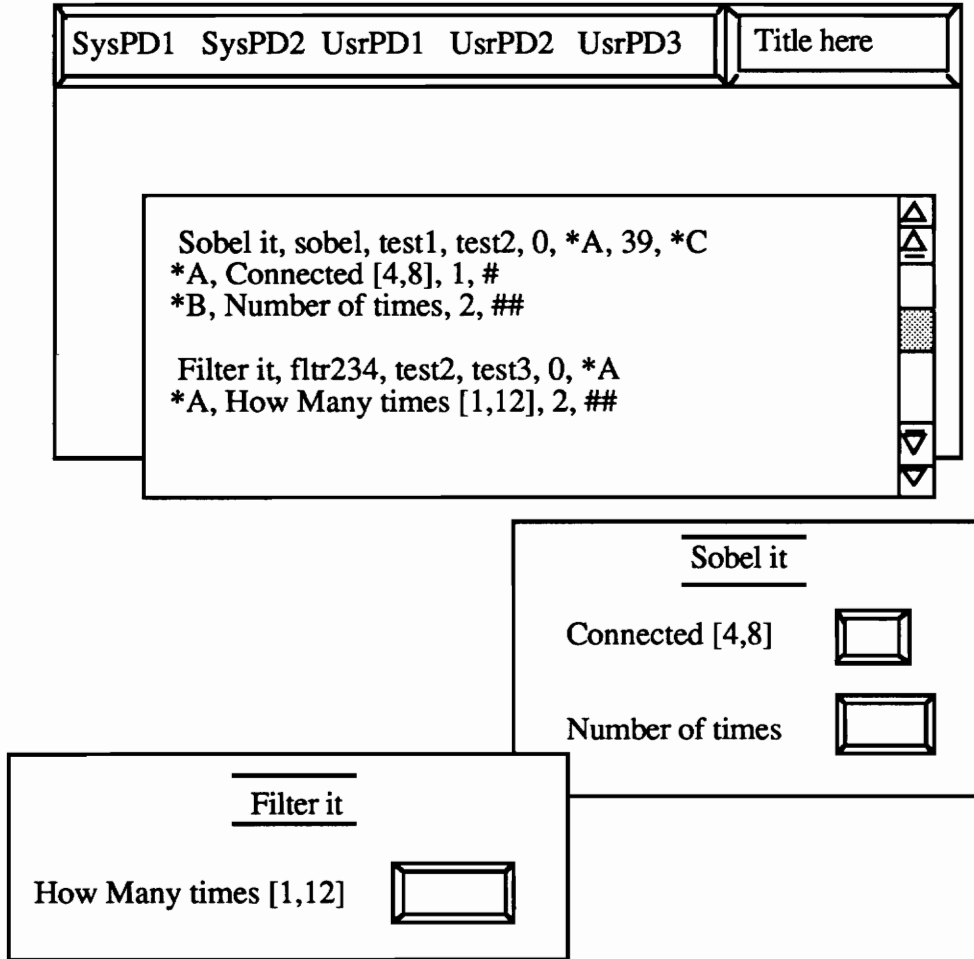


Figure 3: Possible scenario for the user to instantiate modifiable menus.

4.4 Scope of Project

At present, the Gipsy software is not available for the IBM PC, and a full scale implementation of PC-Gipsy, as described above, is well beyond the scope of this project. However, as a first version, an interface and application has been designed to allow users to view images generated by Gipsy. These files are referred to as standard image files (SIF). A user

working offsite could log into Gipsy via either a network connection on the mainframe or via a modem and proceed to download a SIF file to a personal computer. The user can also write SIF files, as well as read and write binary (raw 8-bit image data) files.

The interface has been designed to allow the user to view multiple downloaded SIF images and scroll through those images on both the horizontal and vertical planes. The user can also move, resize, and select windows. Within each window, the user has several commands, including:

- 1) Fatbits display
- 2) Histogram display
- 3) Subimage acquisition
- 4) X,Y coordinate of any pixel

For options 1, 2, and 3, the user selects an area on the image to view. The fatbits command allows the user to zoom-in on a selected region, while the histogram command allows the user to view the frequency of greyvalues. The subimage acquisition command takes the selected region and creates a subimage at a specified sampling rate. Option 4 allows the user to determine the coordinates of any pixel in the image.

5 Tools

Serious consideration must be given to the tools available for developing a suitable interface for a given application. In the IBM PC market this becomes a predominant theme due to a lack of standards and the limitations of MS-DOS environments. The main considerations include memory, programming language, compiler, linker, DOS extender, and the graphical user interface.

5.1 DOS Extenders

The largest constraint in the MS-DOS environment is the availability of RAM memory. The structure of PC-Gipsy will require that several tasks reside in memory at any given time. This includes the Gipsy kernel, the graphical user interface, and a spawned application. This does not include the memory requirements for storing typical digitized images in RAM memory. For example, a typical image with a size of 512x512 pixels and 256 grey levels would require 262 kilobytes of memory. An equivalent image in color would require 786 kilobytes of memory, well beyond the modest capabilities of DOS.

The DOS operating system has been designed to operate in real mode, which emulates an 8086 processor with a 16-bit address space. This fact restricts memory access to below 640 kilobytes of RAM. By making use of 32-bit addressing available in 80386 environments, a protect mode

memory environment is possible that allows access to several megabytes of RAM.

The combination of system tools chosen for the PC-Gipsy system include:

- Metaware's High C 386 compiler
- PharLap's 386|ASM/Link
- PharLap's 386|DOS - Extender

The High C 386 compiler makes use of the 32-bit environment and the extended instruction set that 80386 machines offer. The result is use of protected mode 32-bit address space, reduced amount of generated code, and improved processing speed. The 386/DOS extender provides the programmer with a layer of abstraction above the real mode of DOS. The programmer does not have to consider the 64 kilobyte segment structure but rather has a flat address space of up to four gigabytes using 32-bit addressing. This actually gives both a run-time speedup in processing, as well as a reduction of code size. The extender does not change the applications DOS or Bios calls, but rather intercepts them and passes them on to DOS in real mode. Using the extender, a programmer can also mix real and protected mode code and spawn both real and protected mode programs.

For the first version of PC-Gipsy, the Microsoft C compiler/linker was used because the libraries used to generate the interface were not available for a protected mode extended memory. A beta version of the graphical user interface library/toolkit has been created at the time of this writing, and a fully operation version is expected in Spring, 1991.

5.2 Graphical User Interface Toolkits

A GUI toolkit is a software package that allows a programmer to build a graphical user interface (GUI). A typical GUI toolkit will allow a

programmer to create pulldown menus, popup menus, windows, buttons, icons, and other objects associated with a direct manipulation environment [Shneiderman, 1987]. The term graphical user interface toolkit is a misnomer, however. The interface objects it creates can be represented either textually or as bitmapped images. The textual representation uses the character as the smallest identifiable object on the screen. The bitmapped representation uses the pixel as the smallest discrete object. GUI toolkits are generally designed to produce either text-based or bitmap representations, but not both.

There are tradeoffs between the two that can affect the choice of either a text-based GUI or a bitmapped GUI. Textual GUI's are generally much faster than their bitmapped counterparts and require far less random access memory to operate. They are more limited in terms of functionality and flexibility, and visually these have a limited appeal. On the other hand, bitmapped GUI's are more flexible and more functionally robust. For example, they are capable of displaying images and performing animation. They also give the user readily identifiable iconic cues. Unfortunately, bitmapped GUI's require significantly more computer resources, particularly random access memory and processing capabilities.

The process by which an individual chooses a GUI package is not currently based upon objective information, but rather on some combination of the following:

- system requirements
- anecdotal information (e.g., recommendation)
- previous experience
- cost
- distribution fees
- response speed
- who has a strong market share
- advertisements
- amount of documentation

- blind luck

It is important to consider the fact that GUI toolkits are generally very complex and difficult to learn. There is usually a considerably high learning curve associated with GUI toolkits. Even after a package is learned, it usually takes a considerable programming effort to produce a user interface. Because of the high cost in terms of programming hours, a software developer must strongly consider the choice of GUI toolkits.

Choosing a package based upon minimizing development time while retaining flexibility and functionality is a rather involved issue. A package can be made easy to use, but it may not be functionally robust enough to satisfy the needs of the application. A package that is considered highly robust may require several months to learn and still require an intensive programming effort. An objective means of comparing one GUI toolkit with another would be highly desirable. No such methodology exists, and it may prove to be a very difficult task to produce one.

The package that was chosen for this project is Menuet by Ithaca Street Software. Menuet was chosen specifically because of its zero distribution fees, its promise to support protected mode memory under MS-DOS, and its sizable set of attributes. An assessment of programming difficulty could not be determined at time of purchase. See Appendix 8.1 for a sampling of other DOS-based GUI toolkits available on the market.

Menuet at its most basic level is a set of graphical user interface objects. These objects include:

- Text Items
- Buttons
- Icon Fields
- Slots
- Knobs
- Gauges

- Arrows

Each object has a set of attributes which affect both the appearance of the object (i.e., color, shape, border) and the interactive characteristics. For example, buttons can be given a certain color and border style and be made to behave like check buttons. A set of objects is grouped to become a form. A form also has a set of attributes which affect its appearance. There are several form types, which include:

- Horizontal Bar Menu
- Pull-Down Menu
- Icon Panel
- Button Panel
- Dialog Box
- Check Button Form
- Edit Form
- List Form
- Check List Form
- Indicator Form

Running the interface is done by form processing functions, which do such mundane things as highlight and un-highlight buttons, as well as return an index number to the item that was selected by the user. The form processing functions also provide mouse and keyboard support.

In Menuet there is also the concept of a window. When an application is first initialized and opened, a default window (window 0) is created and displayed. This default window has a title, and contains a menubar form. Up to seven additional windows can be created, displayed, closed, and destroyed. Each additional window also contains a title and menubar.

Menuet is built upon the graphic package called MetaWINDOW by Metagraphics Software Corporation. The package, or toolkit as it is called,

provides a basic graphics library, run-time system support for graphics hardware, virtual bitmaps, and window capability. MetaWINDOW is a graphics package that has a strong market share, and it is used by many of the GUI developers in the IBM PC domain.

6 Development Issues for PC-Gipsy

With the development of PC-Gipsy, several interface and system issues arose that had direct impact on the final design of the system. These issues were related to the hardware on which PC-Gipsy was platformed, the GUI toolkit used to program the interface, and the structure of the file format used to represent images in Gipsy.

6.1 IBM PC/AT Platform

The IBM PC/AT platform was chosen for PC-Gipsy because of its accessibility by a wide audience of users. The prices for PC-clones are well within the range of an average user, and the performance levels for the new machines have made image processing a viable option. The ability to display digital images has also improved with addition of VGA and SuperVGA graphics. For example, SuperVGA can operate at a resolution of at least 640x480 with 256 colors, which is sufficient to display many types of images. The flexibility in configuring the hardware in the PC environment also offers users the option of integrating image processing hardware into their system.

However, there are drawbacks which must be considered. First, SuperVGA is not a graphics standard and comes in several screen resolutions that can vary from one manufacturer to another. For the programmer this becomes a concern because different screen resolutions

can affect the general look and feel of the interface. Also, SuperVGA can only represent 64 grey levels. Although perceptually 64 grey levels may be sufficient, most grey level images have 256 grey levels, and those ought to be displayable. It would also be useful to have a greater number of available colors.

Second, there is the issue of portability in an environment where there exists a multitude of system configurations. The systems for which PC-Gipsy is destined will be required to meet a minimum level of performance and storage specifications. This includes having an 80386-based machine with several megabytes of random access memory. Without these minimum requirements, protected mode memory is impossible.

6.2 Window Design Issues

One main distinction of the Menuet window versus many other windowing systems is the use of pulldown menus in all subwindows. Figure 4 shows an example of a generic Macintosh style window and the default window style produced by Menuet. This leads to some interesting design issues for assigning functions. A designer must decide which commands need to be available within the secondary window and which should be available only in the main window. This forces the designer to develop some conceptual distinctions between commands. One effect could be a simpler menu system, where a small cohesive subset of commands become associated with an individual secondary window and a set of master commands are associated with the main window. However, if such a conceptual distinction cannot be made, the interface may prove clumsy to operate. For example, if there exists a set of commands for manipulating a bitmap image, with some commands only operating on a portion of the bitmap and others operating on the whole bitmap, the user may end up selecting a command that affects the whole image rather than a preselected region. Also, if the set of commands becomes excessive, then there will exist a space problem with the typically smaller secondary window. A

scroll feature in the pulldown system becomes essential, and the number of slots will vary given the size of the window.

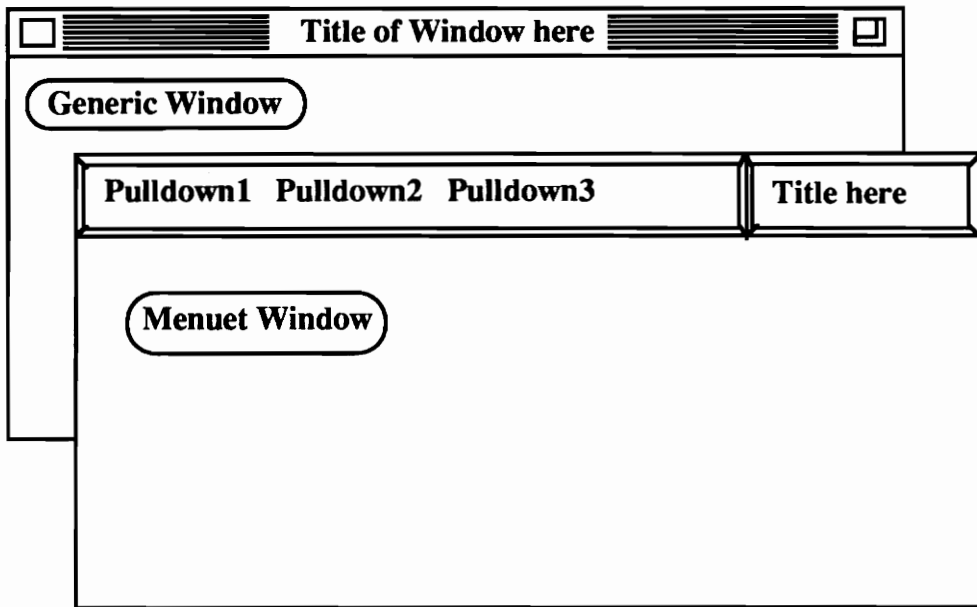


Figure 4: A generic style window style and the default window style in the Menuet graphical user interface.

In addition to the menubar, another distinction of Menuet from other DMI type interfaces is the means by which the user selects an item from the menubar. In the Macintosh interface, for instance, the user opens a pulldown on the menubar by a mouse-down command. By moving the cursor over the pulldown menu, the user can select a command by a mouse-up event. For Menuet, however, the user must use a full mouse-click to open the pulldown and then yet another key click to select a command.

The windowing system supplied by Menuet allows a programmer to create and display a window easily. This window contains a menubar and a title. No other attributes are associated automatically with a window. Windows in systems like Apple's Macintosh typically have scrollbars available to scroll both horizontally and vertically. These window styles also have the ability to be resized and moved. In the Macintosh window,

the titlebar is associated with the ability to move the window, and a small button at the apex of the horizontal and vertical scrollbars (Figure 5) is the resize function. The user interaction that occurs can be described by using User Action Notation [Siochi, 1989]. In the Macintosh-style dialogue for resizing or moving a window (Table 1), a methodological equivalence can be observed.

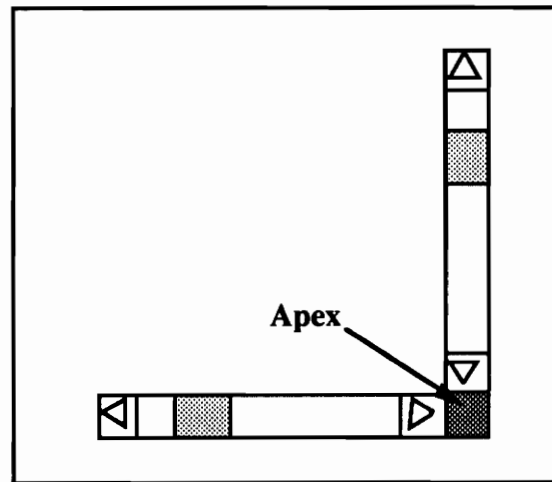


Figure 5: The apex of the vertical and horizontal scrollbars. In a Macintosh-type window this area is reserved for the resize window function.

Task: Resize Window		Macintosh
User	Feedback	System
~[Resize btn]Mv ~[x,y]* M^	active_window! outline of window is resized with change of cursor position resized_active_window -! Redraw Window	

Task: Move Window		Macintosh
User	Feedback	System
~[Titlebar]Mv	active_window!	
~[x,y]*	outline of window is moved with change of cursor position	
M^	active_window -! Redraw Window	

Table 1: The UAN description of human-computer dialogue for moving window and resizing a window in the Macintosh-style interface.

For Menuet, however, the two functions are methodologically different (Table 2). When a user chooses to resize a window, the user first mouse clicks on a button or pulldown menu selection that calls this function. The system then erases the window and draws an outline of the window. This outline contains a small button in the lower right corner. The user must then do a mouse down on this button for the purpose of resizing the window. A mouse up locks the outline in place. The user must then click once again to draw the resized window. The window move function, however, requires the user to mouse down on a button or pulldown menu selection to initiate the move function. The user then can move the window until a mouse up event occurs and the window is redrawn. First, the resize window function is far more complex, requiring three full mouse clicks to move a window from one position to another. Second, there is the problem that after using the resize function, the user may issue a full mouse click when requesting a move operation, which would be considered by the system to be a completed sequence of events.

We must strongly consider this level of detail given that the resize and move functions may be used quite frequently by users. The feedback mechanism will make the distinction obvious to the user, but frequent user errors may occur. The author made the decision to distinguish the two functions both spatially and conceptually. A move button was created at the

apex of the horizontal and vertical scrollbar. This is of course associated with the resize button found on Macintosh interface. It was felt that this would not become an issue given the adequate feedback mechanism that exists. It was also felt that the move function would be utilized more frequently than the resize function and needed to be more accessible. The resize function was placed as an item in the window's pulldown menu. This proved to be a more consistent place for this function since selecting items in the pulldown menu system required a full mouse click to activate.

Task: Resize Window		Menuet
User	Feedback	System
~[Resize btn]Mv^	active_window! outline has a small button (R_btn) on lower right hand side	
~[R_btn]Mv	outline of window	
~[x,y]*	window is resized with change of cursor position	
M^	outline is fixed to new size	
Mv^	resized_active_window -! Redraw Window	

Task: Move Window		Menuet
User	Feedback	System
~[Move btn]Mv	active_window!	
~[x,y]*	outline of window is moved with change of cursor position	
M^	active_window -! Redraw Window	

Table 2: The UAN description of human-computer dialogue for moving window and resizing a window in the Menuet-style interface.

6.3 Standard Image File (SIF) structure

The SIF format is the generic format for all images in Gipsy. This format can support images of any size, resolution, and data type.

PC-Gipsy's internal data structure will be based on this image file format, thereby allowing easy access to images produced by Gipsy.

The SIF structure is broken down into four major components:

- 1) Initial Record
- 2) Identification Record
- 3) History Record
- 4) Image Data

The initial record contains the record length in bits. This information is used to determine the beginning of the identification record. The identification record specifies the number of history records and the format of the image data. All records within a SIF file are of equal length.

The image data are compressed to make efficient use of non-volatile memory, yet suffer a minimum loss of information. The data are compressed by examining the minimum and maximum data values and storing the data using a minimum number of bits to represent the data. The information required to determine this encoding includes:

- nbits - number of bits per picture element
- mode - data mode
- nbnds - number of bands

Gipsy compacts integer data based on the range of image data values (i.e., idmin, idmax). The Gipsy system allocates a minimum number of bits per picture elements depending upon the number of bits required to represent the full range of possible values. These integer data can also be formulated in 2's complement for the purpose of representing negative numbers. The mode indicates the existence of 2's complement data encoding.

For most SIF files, one record represents one line (row) of image data. Therefore, the number of records allocated for data is also the number of rows in a given image. For multiband images, several records can represent one line of image data. For example, an RGB image has three bands, each band representing one of the primary colors. Gipsy will store them as three sequential records assuming one record is equivalent to one line of image data. The total number of records (i.e., rows) stored for the image will therefore be the number of rows in the image times the number of bands. The exception occurs when the image becomes large enough to warrant the system to split a line of image data into multiple records. This format will not be discussed, but for further information the "Gipsy User's Guide" is recommended.

For the purpose of displaying images in the user interface, an Internal Standard Image File Structure (ISIFS) was created which maintains both the data and the SIF structure. The ISIFS has an initial record structure, an identification record structure, a pointer to the history records, and a pointer to the data (i.e., picture elements). At present, the first four bytes of each record (part of the VAX record structure) are stored. However, some changes have been made. First, the initial record and identification records do not store the data padding. The history records are simply recorded as character data. The image data are converted into one of three formats:

- 1) unsigned char (1 byte)
- 2) signed short (2 bytes)
- 3) float (4 bytes)

Two issues must be addressed given the conversions which will occur when loading a SIF image file. First, some of the conversions into float do cause a loss of precision. The user must be informed of this loss of precision when it occurs. Secondly, the current system cannot convert the floating point format that is produced by the VAX into the floating point format used the PC. Until this is resolved, various types of SIF images cannot be loaded into the internal image format used by PC-Gipsy.

7 The Final Product

The final product of this project consists of one main window and several secondary windows. The commands in the primary window have the primary function of creating secondary image windows and displaying images within them. The secondary window has scrollbars which allow the user to scroll an image both vertically and horizontally. Both are described below in detail.

7.1 The Main Window

The main window (Figure 6) has a menubar with two pulldown menus, "Files" and "Windows". A third selection on the menubar is a "Quit" command which terminates the application. The "Files" pulldown menu has four commands which allows the user to:

- Open - open and display a SIF file
- Close all - close all open secondary image windows
- Import - import images from other file formats
- Export - export images for other file formats

Selecting an "Open" command causes a scrollable list form to appear (Figure 7) that contains a listing of all files with a ".SIF" extension. The user can choose one of the files or choose files from a different directory. Once a file is selected, a secondary window will open and an image is loaded into the window bitmap. The "Close all" command will close all the

secondary windows which have not been closed. The "Import" command opens a form (Figure 7) which allows the user to select from different types of files to import into the system. Once the user selects a file type to import, the user can then select a file. A secondary window will then be created, and a window bitmap is loaded with the imported image. The "Export" function essentially reverses that process. The "Windows" pulldown is an alternative means for the user to select a window. Those windows which are not open will have their names ghosted (un-highlighted).

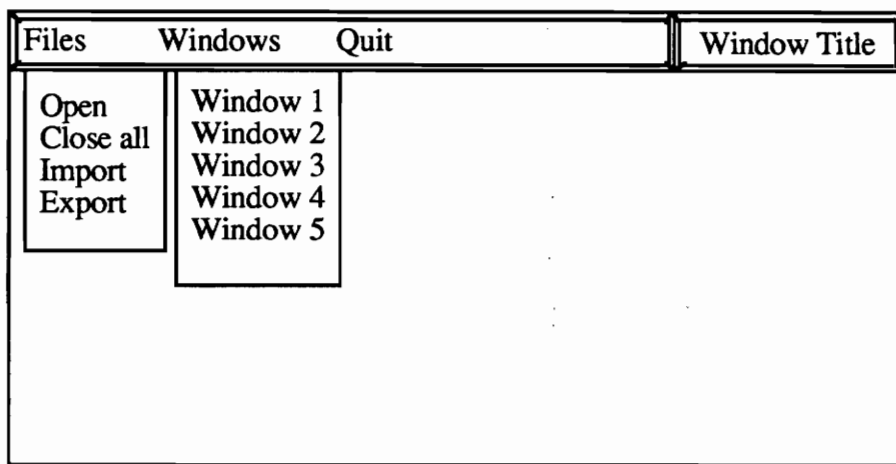


Figure 6: The main window and its pulldown menus.

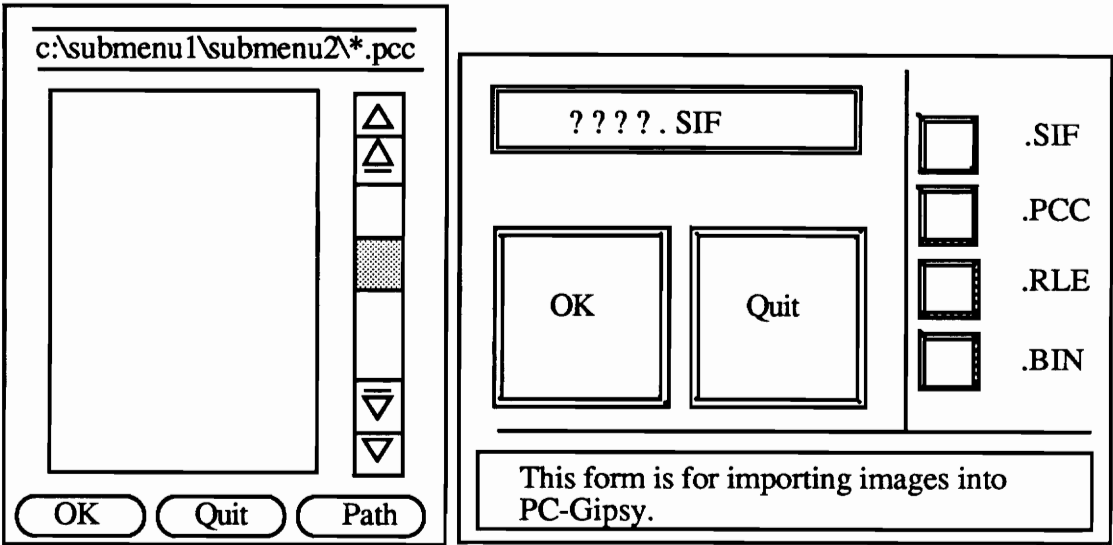


Figure 7: The file list form for selecting SIF files, and a form for importing images.

7.2 The Secondary Image Window

The secondary window (Figure 8) is significantly more complicated with a vertical scrollbar, horizontal scrollbar, a move button, and an area for displaying the coordinate of a given pixel on an image.

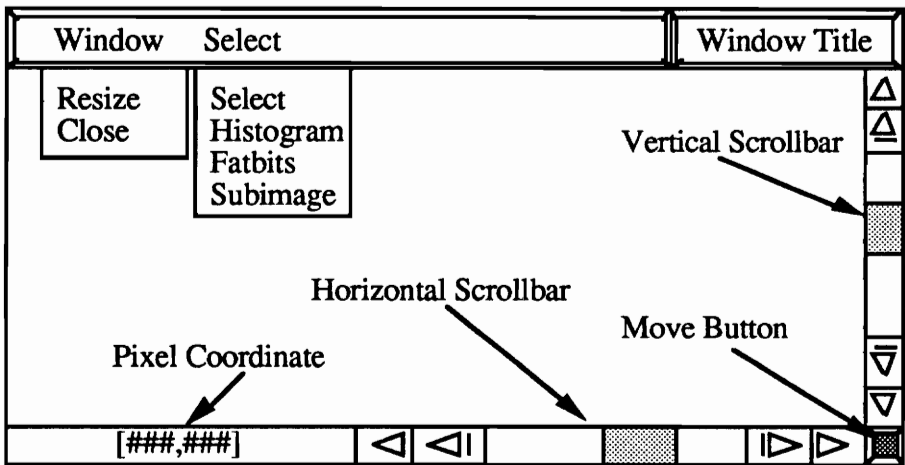


Figure 8: Example of a secondary image window for PC-Gipsy.

Each scrollbar has four buttons, each with an arrow to represent the scroll direction. The arrows with an extra line parallel to one side indicate a scroll to the extremes of the image. On the far lower right hand corner of the window is a move button which allows the user to move the entire window to a different position on the screen.

The image window has a set of commands associated with it. These commands are “local” in the sense that they perform actions directly relevant to the window. Under the “Window” pulldown, the user can choose to resize the window. Under the “Subimage” pulldown the user can:

- Select subimage - select a rectangular region on the image
- Histogram subimage - display a histogram of the selected region
- Fatbit subimage - do a zoom of the selected region
- saVe subimage - save a sampled image of the selected region

The “Histogram” command creates a histogram (Figure 9) of the pixel data within the selected subimage. These data are scaled to meet the dimensions for the form displaying the histogram. The “Fatbit” command zooms and displays the selected area, and allows the user to determine the x,y location of any given point in the fatbit image. The “saVe” command allows the user to write a binary file with a user-specified sampled set of image data.

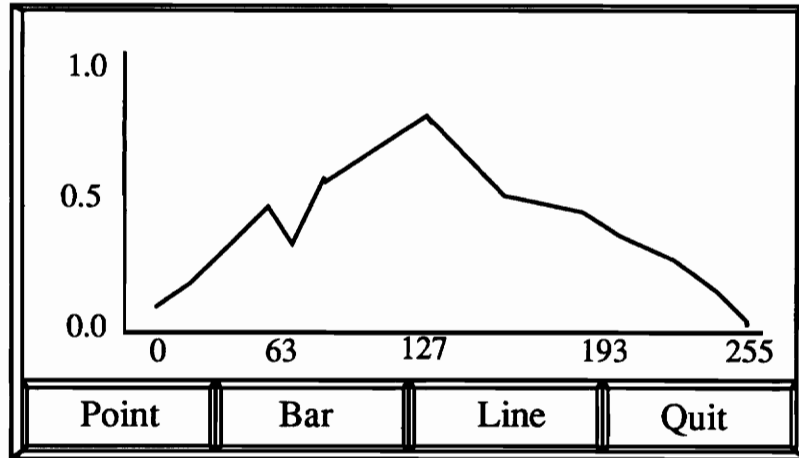


Figure 9: The histogram form with options to view data in various ways.

7.3 The Window Manager

The window manager allows the user to choose a window by simply pointing at it with a mouse and clicking. Once a window is selected and made active, the user can manipulate or select objects in the form associated with the window. The user can also choose to click in yet another window, thereby establishing it as the active window. Only one active window can exist at a time. It is possible for a window to be obscured by other windows. If such a situation occurs, the user can return to the main window and select the “Window” pulldown menu. This pulldown menu contains a list of all the windows a user could create within a given application. Windows which have not been created have their names ghosted (Figure 10) and cannot be selected by the user.

Files Windows Quit			Window Title
Open	window 1		
Close all	window 2		
Import	window 3		
Export	window 4		
	window 5		

Figure 10: Example of ghosting in “Windows” pulldown. Slots 3 and 4 are ghosted.

8 Conclusion

8.1 Design Conclusions

1) A direct manipulation interface is a viable option for the PC platform. An interface for the PC can have many of the state-of-the-art characteristics associated with contemporary user interfaces. This includes the ability to display and manipulate bitmapped images generated by other sources.

2) By both the virtue of being a direct manipulation interface, the overall usability of Gipsy will improve. Also, the ability to view and manipulate Gipsy images from a remote access point (i.e., PC) will improve the accessibility of the system to a broader population of users.

3) It is important to consider carefully the choice of GUI toolkits. The considerations include:

- programming effort to create an user interface
- ability to display bitmapped images
- ability to operate in protected mode memory
- configured for various high resolution graphic drivers

4) In designing a system, it is also necessary to consider the platform on which the system is being designed. The PC platform comes with a wide range of choices in terms of performance and capabilities. A direct manipulation environment, however, requires lots of memory and fast graphics. Also, high resolution displays with lots of colors and required. The minimum specifications for a PC platform are as follows:

- PC 80386 machine
- SuperVGA with 256 colors
- Several megabytes of dynamic memory

8.2 Future work

1) First and foremost is a user defined menu system. This menu system must provide access to Gipsy commands as well as the ability for the user to spawn other programs.

2) Need to establish a communication network between the PC and the VAX for the purpose of transferring image files, and for issuing Gipsy commands. The will provide a transition system until Gipsy can be fully ported to the PC environment.

3) The floating point encoding in the VAX is different from the encoding used on PC. The SIF files produced by Gipsy on the VAX may have the image data stored a floating point number. For PC-Gipsy to be able to read all type of SIF files, the issue of being able to convert the VAX floating point encoding to the encoding used on the PC must be resolved.

4) The version of PC-Gipsy was compiled under Microsoft C in real mode because of the unavailability of libraries (Menuet toolkit) which can operate in protected mode. As soon as such libraries are available, the software must be converted to operate in protected mode.

Bibliography

1. R. W. Ehrich, Personal Conversations at Virginia Polytechnic Institute and State University, Department of Computer Science, 1990.
2. E. Garland, R. W. Ehrich, "A Gipsy Primer", Virginia Polytechnic Institute and State University, Spatial Data Analysis Laboratory, 1986.
3. B. W. Kernigan, P.J. Plauder, "Software Tools", Addison-Wesley Publishing, Reading Massachusetts, 1976
- 4 K. H. Melder, "The Application of Task Analysis for Interface Design of Image Processing systems", Unpublished document produced at Virginia Polytechnic Institute and State University, 1990.
- 5 Microsoft Corporation, "The Benefits of the Graphical User Interface: A report on New Primary Research", Prepared by Temple, Barker & Sloane, 1990.
6. K. R. Paap, R. J. Roske-Hoftstrand, "Design of Menus", in Handbook of Human-Computer Interaction, North-Holland, pp. 205-235, 1988.

7. B. Shneiderman, "Designing the User Interface: Strategies for Effective Human-Computer Interaction", Addison-Wesley Publishing Company, Reading Massachusetts, 1987.
8. A. Siochi, D. Hix, H. R. Hartson, "The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs", Virginia Polytechnic Institute and State University, Department of Computer Science, 1990.
9. M. R. Wade, R. W. Ehrich, "Gispy User's Guide", Virginia Polytechnic Institute and State University, Spatial Data Analysis Laboratory, 1987.
10. J. E. Ziegler, K. P. Fahnrich, "Direct Manipulation", in Handbook of Human-Computer Interaction, North-Holland, pp. 123-174, 1988.

Appendix

Graphical user interface libraries/toolkits for the IBM PC

<u>Toolkit Name</u>	<u>Company</u>	<u>Telephone No.</u>
Aewindows		
"C"erious Tools	TRS System Ltd	(516) 331-6336
C-Scape	Oakland	(800) 233-3733
Chatterbox	Courseware Appl.	(217) 359-1878
Curses	Aspen Scientific	(303) 423-8088
C-Worthy	Solution Systems	
ESIX	Programmer's Odyssey	(800) 848-8731
Flash Tools!	Avis Hill	(818) 709-8366
Formation Desktop D.	Aspen Scientific	(303) 423-8088
GFX Fonts and Menus	C Source	(816) 353-8808
Graphics-MENU	Island System	(617) 273-0421
Hi-Screen XL		(800) 338-2852
Hyperinterface	Avanpro	(213) 454-3866
JAM & JAM/DBi	JAYCC	(800) 458-3313
Menuet	Ithaca Street Software	(303) 494-8865
Mewel 3	Magma Software Sys.	(201) 912-0192
Panel plus II	Roundhill	(201) 569-2265
PC/Forms	Golden Solutions	(800) 338-6754
Say What?!	Software Science	(415) 467-7077
Screen Generator	West Chest Group	(215) 644-4206
Screen-Flex	Info0Flex	(415) 340-0220
Screen maker Pro		(814) 234-8088
Terminal Control	MainStream Software	(214) 250-3813
T-Windows	Oxford Consulting	(800) 962-7338
User Interface Library	Software Artistry	(317) 876-3042
Vermont Views		(800) 848-1248
Vitamin C 3.2		(214) 416-6447
vLIB	Pathfinder Assoc.	(408) 984-2256
WNDX	WNDX	(800) 667-9639
Zmenu/ZView	Data Mgmt.Consults	(800) 423-0930

The PC-Gipsy Source Code

```
cl /c /AL %1.c
```

```
cl main13 import3 binform imgload3 wn2manage selectfl rsif13 histform /link  
met_bd1d mw_d1dmc menuetml mnutilml /ST:4000
```

```

/* ----- */
/*

```

Module Name: MAIN13.C

Purpose: Main program for PC-Gispy

Procedures:

```

load_SIF_file()      - load SIF image
load_PCX_file()      - load image produced
                      by paint packages
load_BIN_file()      - load a straight binary
                      file
init_all_windows()   - initialize all the
                      windows for look and
                      feel
main()               - main entry point

```

Description: The is a functional prototype of PC-Gispy. Most of the windowing components have been implemented, as well as supplying form for allowing the user to change parameters.

A program that allows SIF files has been included, but still needs to be integrated into the main program. Please look at the code to read binary files as guidance to do this.

Hardware: Compaq Despro 386/20

Compiler: Microsoft C version 5.1

```

Linker:      cl /c /AL %1.c
             cl main binform /link met_bd1d mw_d1dmc
             menuetml mnutlml /ST:4000

```

Requires MetaWindows 3.6 and
Menuet 1.7E

Author: Karl Melder - Fall 1990

```

/* ----- */
/*

```

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

```

```

#include <GRconst.h>
#include <GRports.h>
#include <GRextrn.h>

#include <menuet.h>
#include "project.h"
#include "gipsy.h"

#define FAR_ALLOC(n)      _fmalloc(n)
#define FAR_FREE(p)      _ffree(p)

#define GR_DEVICE   VGA640x480X      /* Graphics Device -
                                      SuperVGA, 256 color */

/*#define GR_DEVICE   VGA640x480*/    /* SuperVGA, 4 color */

/* #define GR_DEVICE   VGA320x200 */  /* VGA 256 color */

char fnt_name[80];      /* A smaller additional
                        font for tight spots */
char *fnt_buf;         /* Used to set temp. font */
long fnt_size;         /* Used to set temp. font */

#define IMGWNDW_CNT 8  /* windows 0 thru 7 */

static IMAGEWINDOW      wninfo [IMGWNDW_CNT]; /* Structure for each
                                                window - has info on
                                                view window size and
                                                has image bitmap */

static ISIFS            asimage [IMGWNDW_CNT]; /* Structures for
                                                Internal SIF for each
                                                SIF image opened. */

/* ----- */

#define nMAIN 3          /* # of items in the main menubar */

/* object descriptors define individual menu items */

ob_desc mainITEMS[nMAIN] =
{
    { 0, 0, 0, 0, 0, 0, 0, 0, 'F', 0, " Files",  NULL, 0, 0, 0, 0, NULL },
    { 0, 0, 0, 0, 0, 0, 0, 0, 'W', 0, " Windows", NULL, 0, 0, 0, 0, NULL },
    { 0, 0, 0, 0, 0, 0, 0, 0, 'Q', 0, " Quit ",   NULL, 0, 0, 0, 0, NULL }
};

/* form descriptor defines the object group as a discrete form */

```

```

fm_desc mainMENU =
{
    0, 0, 0, 0, 0, 0, 0, NULL, 0, 0, 0, 0, nMAIN, mainITEMS,
    0, NULL, 0, NULL
};

/* ----- */

#define nSUB1 4

ob_desc sub1ITEMS[nSUB1] =
{
    { 0,0,0,0,0,0,0,'O',0," Open ",NULL,0,0,0,0,NULL},
    { 0,0,0,0,0,0,0,'C',0," Close ",NULL,0,0,0,0,NULL},
    { 0,0,0,0,0,0,0,'I',0," Import ",NULL,0,0,0,0,NULL},
    { 0,0,0,0,0,0,0,'E',0," Export ",NULL,0,0,0,0,NULL}
};

fm_desc sub1MENU =
{
    0,0,0,0,0,0,0,NULL,0,0,0,0,nSUB1,sub1ITEMS,0,NULL,0,NULL
};

#define nSUB2 5

ob_desc sub2ITEMS[nSUB2] =
{
    { 0,0,0,0,0,0,0,'1',0," Window 1 ",NULL,0,0,0,0,NULL},
    { 0,0,0,0,0,0,0,'2',0," Window 2 ",NULL,0,0,0,0,NULL},
    { 0,0,0,0,0,0,0,'3',0," Window 3 ",NULL,0,0,0,0,NULL},
    { 0,0,0,0,0,0,0,'4',0," Window 4 ",NULL,0,0,0,0,NULL},
    { 0,0,0,0,0,0,0,'5',0," Window 5 ",NULL,0,0,0,0,NULL}
};

fm_desc sub2MENU =
{
    0,0,0,0,0,0,0,NULL,0,0,0,0,nSUB2,sub2ITEMS,0,NULL,0,NULL
};

/* ----- */

static int    load_SIF_file();
static int    load_PCX_file();
static int    load_BIN_file();
static void    init_all_windows();
void          main();

```

```

void main( int argc, char *argv[] )
{
    int i,                /* Counter */
        done;            /* Escape from while loop */

    char msg[80],         /* Message to the user */
        flname[30];      /* file name */
    int fl_type;          /* file type (e.g., PCC) */

    int ret,              /* General return value */
        new_window;      /* The index # of the new window */

    int pick,             /* Users selection from menu */
        mainpick,        /* Users selection on menubar */
        subpick,         /* Users selection on menubar pulldown */
        subsubpick;      /* not used */

    int current_act_wh,   /* current active image window */
        selected_win;

    unsigned
    int greyvalue;        /* grey value for color palette */

    char stuff[8];
    char *targv[3];       /* input variables */
    int targc;           /* input variables */

    attr_blk *attrptr;    /* attribute block */
    point *mouse_click;   /* location of mouse click */

    palData *Mono;        /* Used to set Palette */
    palData mono;         /* palette array */

    FILE *stream;         /* file pointer */

    /* Initialize graphics Mode - must have 256 colors */

    if( GR_DEVICE == VGA640x480X )    /* SuperVGA 256 colors */
    {
        strcpy(stuff, "/U:5");
        targv[1] = stuff;
        targc = 2;
        SetMnFontName("M8_FONT.ISS");
    }

    else if( GR_DEVICE == VGA320x200 ) /* VGA 320x200 256 colors */
    {
        strcpy(stuff, "/E:8");
        targv[1] = stuff;
    }
}

```

```

        targc = 2;
        SetMnFontName("SYSTEM07.FNT");
    }

    else if( GR_DEVICE == VGA640x480 )
    {
        strcpy(stuff, "/U:1");
        targv[1] = stuff;
        targc = 2;
        SetMnFontName("\\menuet\\fnt\\BLD013.FNT");
    }

/* Load up the secondary font for the windows */

    if( GR_DEVICE == VGA640x480X )
        strcpy(fnt_name, "\\menuet\\fnt\\S8_FONT.ISS");
    else if ( GR_DEVICE == VGA320x200 )
        strcpy(fnt_name, "\\menuet\\fnt\\BLD008.fnt");
    else if ( GR_DEVICE == VGA640x480 )
        strcpy(fnt_name, "\\menuet\\fnt\\BLD008.fnt");

/* make sure font exists and determine it size */

    fnt_size = FileSize( fnt_name );
    if( fnt_size < 1 )
        TermApp("Could not locate sample MW font");

/* allocate a buffer for the font - must set memory to use under 640K */

    SetSvMethod(svMEM);                /* Make sure we alloc within 640K */
    fnt_buf = (char *) FAR_ALLOC( fnt_size+8 );
    SetSvMethod(svANY);

/* load the font */

    if( LoadDiskFile(fnt_name, fnt_buf ) != 0 )
        TermApp("Error loading font");

/* Initialize the application - the arg list is same as would
normally be passed to GrQuery. */

    InitApp( targc, targv );

/* Customization of Windows LOOK and FEEL */

```

```
ReadAttrTbl("PROJECT.ATT");
init_all_windows( );
```

```
/* Open the application and window 0, giving app name and a main
   menu pointer. Start up the message queue, which starts MW's event
   queue - nonzero arg -> show mouse cursor. */
```

```
SetWnBdrThk(2);
SetWnShwThk(0);
```

```
OpenApp( "MnSamp02", &mainMENU );
OpenMsgQueue( 1 );
```

```
/* Lets set the color palette */
```

```
if( GR_DEVICE == VGA640x480X )
{
    greyvalue = 0;
    for (i=192; i<256; i++)
    {
        mono.palRed   = (unsigned short) greyvalue;
        mono.palGreen = (unsigned short) greyvalue;
        mono.palBlue  = (unsigned short) greyvalue;
        WritePalette(0,i,i,&mono);
        greyvalue = greyvalue + 1024;
    }
}

if ((stream = fopen("palette.txt","w")) == NULL)
    NotifyUser(-1,-1,"DEBUg","Could not open file");
for (i=0; i<16; i++)
{
    ReadPalette(0,i,i,&mono);
    fprintf(stream,"palette[%d] R:%u G:%u B:%u\n",
            i,
            mono.palRed,
            mono.palGreen,
            mono.palBlue);
}

fclose(stream);
```

```
/* Initialize the menu/submenu tree */
```

```
/* main menu item 1 */
```

```

InitMenu(&sub1MENU,ALIGN_VERT);
AttachSubMenu(mainITEMS, &sub1MENU);

/* main menu item 2 */
InitMenu(&sub2MENU,ALIGN_VERT);
AttachSubMenu(mainITEMS+1, &sub2MENU);

/* main menu item 3 */
mainITEMS[2].flags |= IS_EXITABLE;

```

```

/* Make items in Window pulldown not selectable */

```

```

sub2ITEMS[0].flags &= ~IS_SELECTABLE;
sub2ITEMS[1].flags &= ~IS_SELECTABLE;
sub2ITEMS[2].flags &= ~IS_SELECTABLE;
sub2ITEMS[3].flags &= ~IS_SELECTABLE;
sub2ITEMS[4].flags &= ~IS_SELECTABLE;

```

```

/* process the menu */

```

```

done = 0;
while ( !done )
{

```

```

    pick = SelectOb(mainITEMS,nMAIN,0);
    mouse_click = MouseXY(1);

```

```

/* break down return value from SelectOb into numbers
representing the pulldown menu selection. */

```

```

    mainpick = pick & 0x000F;
    if (mainpick == 0x000F) mainpick = -1;

```

```

    subpick = (pick >> 4) & 0x000F;
    if (subpick == 0x000F) subpick = -1;

```

```

/* If the user has requested a quit or escape, then break */

```

```

    if( mainpick == 2 )    /* item 2 = quit */
        break;

```

```

    if( mainpick == 0)      /* make a window and open file */
    {

```

```

        switch (subpick )

```

```

{
    case 0:      /** Open an Image File **/

        /* get a file name of a .SIF file */

        if( (ret = get_file_name(filename, SIF)) == -1 )
            break;

        if( (new_window = load_SIF_file( filename )) <= 0 )
        {
            NotifyUser(-1,-1,"ERROR","This file cannot be load");
            break;
        }

        /* Enter the control loop which allows the to
           scroll the image, and switch between windows. */

        ret = dowind( new_window, wninfo );

        /* if dowind returns a notice of a window close
           un-highlight the slot in pulldown menu */

        if ( ret > 0)
            sub2ITEMS[ret-1].flags &= ~IS_SELECTABLE;

        break;

    case 1:
        /* close_act_img_wh();*/
        break;

    case 2:      /** Import an image **/

        if( (ret = get_import_fn(filename, &fl_type)) == -1)
            break;

        if( fl_type == PCC )
            load_PCX_file( filename );

        if( fl_type == BIN )
            load_BIN_file( filename );

        /*else if( fl_type == MSP )
            load_MSP_file( filename );*/

        break;
}

```

```

        case 3:
            Beep();
            NotifyUser(-1,-1,"Note","Not Yet Implemented");
            break;
    }
}

if(mainpick == 1)
{
    ret = dowind( subpick+1, wninfo );
    if ( ret > 0)
        sub2ITEMS[ret-1].flags &= ~IS_SELECTABLE;
}

/* Check if users has clicked in another window.
   If another window was clicked in, and it was
   not one of the image windows (1 thru 8) then
   assume the user want to go to main window. */

if( PtlInWindow(mouse_click, 0) == True)
{
    /* Check to see if user clicked in any of the
       the image windows */

    for(i=1;i<8;i++)
    {
        if (PtlInWindow(mouse_click,i))
        {
            selected_win = i;
            /*NotifyUser(-1,-1,"DEBUg","found it");*/
            break;
        }
    }

    if( (selected_win > 0) && (selected_win < 8) )
    {
        ret = dowind( selected_win, wninfo );
        if ( ret > 0)
            sub2ITEMS[ret-1].flags &= ~IS_SELECTABLE;
    }
}

}

/* shut down the message queue */

```

```
    CloseMsgQueue();  
  
    /* shut down the application, displaying passed msg; */  
    TermApp( "End of Pgm" );  
}
```

```

void init_all_windows( void )          /* Init all image windows */
{
    int i;                             /* counter */

    for(i=0; i<8; i++) /* go through all 8 windows */
    {
        SetWnTtlColors(i,LIGHTGREY,LIGHTGREY);
        SetWnColors (i,LIGHTGREY,LIGHTGREY);

        SetRect( &wninfo[i].viewA, 0, 0, 0, 0);
        SetRect( &wninfo[i].imagSrc, 0, 0, 0, 0);
        SetRect( &wninfo[i].imagDst, 0, 0, 0, 0);
        SetRect( &wninfo[i].lcl_viewA, 0, 0, 0, 0);
        SetRect( &wninfo[i].slctR, 0, 0, 0, 0);

        wninfo[i].selected = False; /* subimage not selected yet */

        wninfo[i].gbl_dx = 0;
        wninfo[i].gbl_dy = 0;
        wninfo[i].gbl_np = 0;
        wninfo[i].gbl_nb = 0;
    }
}

```

```

int load_BIN_file( char flname[] )
{
    int    xcols,
           yrows,
           new_window,
           ret,
           i,j,k;
    long    header;
    FILE     *imagefile; /* file pointer */
    unsigned
    char     **ptrarray; /* array of pointers */
    unsigned
    char     *charptr;    /* character pointer */
    rect     line;        /* rectangle that is 1 pixel wide */
    long     imbytes;     /* # of bytes in the image */
    image     *imgptr;     /* pointer to the image */
    imageHeader *hdr;      /* pointer to image file header */
    int      mustread;

    bitmap   *bmap;        /* the bitmap holding the image */

    if((ret = get_binary_form( flname,&xcols,&yrows,&header)) == -1)
    {
        SetActiveWn(0,1);
        return(-1);
    }
    SetActiveWn(0,1);

    /* Hide the cursor to prevent smears */

    HideCursor();

    /* Make a window for the image */

    if( (new_window = mkimagewin(flname,wninfo)) == -1 ) return(-1);

    /* Allocate and load image into memory. If it fails, destroy
       window and discontinue. */

    Notify(-1,-1,"Loading, please wait...");

    if((imagefile = fopen(flname,"rb")) == NULL)
    {
        NotifyOff();
        ShowCursor();
    }

```

```

    NotifyUser(-1,-1,"ERROR","could not open file");
    CloseWindow(new_window,1);
    DestroyWindow(new_window);
    SetActiveWn(0,1);
    return(-1);
}

SetSvMethod(svMEM);
ptrarray = (char **) FAR_ALLOC(yrows * sizeof(char *));
SetSvMethod(svANY);
if(ptrarray == NULL)
{
    NotifyOff();
    ShowCursor();
    NotifyUser(-1,-1,"ERROR", "Could not alloc memory: ptrarray");
    fclose(imagefile);
    CloseWindow(new_window,1);
    DestroyWindow(new_window);
    SetActiveWn(0,1);
    return(-1);
}

SetRect(&line,0,0,xcols-1,0);
imbytes = ImageSize(&line);

SetSvMethod(svMEM);
imgptr = (image *) FAR_ALLOC((unsigned short int) imbytes);
SetSvMethod(svANY);
if(imgptr== NULL)
{
    NotifyOff();
    ShowCursor();
    NotifyUser(-1,-1,"ERROR", "Could not alloc memory: ptrarray");
    fclose(imagefile);
    CloseWindow(new_window,1);
    DestroyWindow(new_window);
    SetActiveWn(0,1);
    return(-1);
}

hdr = (imageHeader *) imgptr;
ReadImage(&line,imgptr);

create_lcl_bmap( &wninfo[new_window].imagMap,
                xcols,
                yrows,
                hdr->imPlanes,
                hdr->imBits, 0);

```

```

wninfo[new_window].gbl_dx = xcols;
wninfo[new_window].gbl_dy = yrows;
wninfo[new_window].gbl_np = hdr->imPlanes;
wninfo[new_window].gbl_nb = hdr->imBits;

```

```

SetSvMethod(svMEM);
for(i=0; i<yrows; i++)
{
    ptrarray[i] = (char *) FAR_ALLOC(xcols);
    if(ptrarray[i] == NULL)
    {
        NotifyOff();
        ShowCursor();
        NotifyUser(-1,-1,"ERROR", "Could not alloc mem:ptrarray[i]");
        SetSvMethod(svANY);
        fclose(imagefile);
        CloseWindow(new_window,1);
        DestroyWindow(new_window);
        SetActiveWn(0,1);
        return(-1);
    }
}

```

```

fread(ptrarray[i],1,xcols,imagefile);

```

```

/* Now convert this data byte for byte */

```

```

for( k=0; k<xcols; k++)
{
    ptrarray[i][k] = ptrarray[i][k]/4;
    ptrarray[i][k] = ptrarray[i][k] | 192;
}

bmap = &wninfo[new_window].imagMap;
bmap->mapTable[0]->rowTable[i] = &(*ptrarray[i]);
}

```

```

SetSvMethod(svANY);
NotifyOff();
display_image( new_window, &wninfo[new_window] );
sub2ITEMS[new_window-1].flags |= IS_SELECTABLE;

```

```

ShowCursor();

```

```

}

```

```

int load_SIF_file( char flname[] )
{
    int          nbits,          /* # of bits per pixel */
               npple,
               nlins,            /* # of lines (ie, rows) */
               ncols,            /* # of columns */
               nbnds,            /* # of bands in the image */
               nsbnds,
               mode,              /* the image mode */
               isif_type;         /* the internal SIF type */

    int          ret,             /* return value */
               new_window;        /* index to new window */

    /* Make a window for the image */

    if( (new_window = mkimagewin(flname,wninfo)) == -1 ) return(-1);

    /* Allocate and load image into memory.  If it fails, destroy
       window and discontinue. */

    ret = load_image_setup( flname, SIF, new_window, &wninfo[new_window] );

    if(ret == -1 ) {
        CloseWindow(new_window,1);
        DestroyWindow(new_window);
        SetActiveWn(0,1);
        return(-1);
    }

    /* Display the image that was loaded into
       memory and selected the pulldown slot */

    readSIF(flname, &asimage[new_window], &wninfo[new_window]);

    NotifyUser(-1,-1,"DEBUG","about to display the image");
    display_image( new_window, &wninfo[new_window] );

    ShowCursor();
    sub2ITEMS[new_window-1].flags |= IS_SELECTABLE;

    return( new_window );
}

```

```

int load_PCX_file( char flname[] )
{
    char      msg[80];          /* message buffer */

    int       nbits,            /* # of bits per pixel */
              npple,
              nlines,           /* # of lines (ie, rows) */
              ncols,           /* # of columns */
              nbnds,            /* # of bands in the image */
              nsbnds,
              mode,             /* the image mode */
              isif_type;        /* the internal SIF type */

    int       ret,
              new_window;

    /* Lets now test the queryPCX functions */

    /*ret = get_file_name(flname,PCC);
    if( ret == -1 ) return(-1);*/

    if( (new_window = mkimagewin(flname,wninfo)) == -1) return(-1);

    /* Allocate and load image into memory.  If
       is fails, destroy window and discontinue. */

    ret = load_image_setup(flname,PCC,new_window,
                           &wninfo[new_window]);
    if(ret == -1 )
    {
        CloseWindow(new_window,1);
        DestroyWindow(new_window);
        SetActiveWn(0,1);
        return(-1);
    }

    /* Display the image that was loaded into
       memory and selected the pulldown slot */

    ReadPCX(flname,&wninfo[new_window].imagMap, 0, 0);

    display_image( new_window, &wninfo[new_window] );
    ShowCursor();
    sub2ITEMS[new_window-1].flags |= IS_SELECTABLE;
    return(new_window);
}

```

```

/ *-----* /
/ *

```

Module Name: IMPORT3.C

Purpose: To have the user pick a file type to import
and then have the user select a file name

Procedures:

<code>select_fn ()</code>	- create a form for selecting a file name
<code>mytask ()</code>	- documentation task called from get_import_fn control loop
<code>get_import_fn()</code>	- get import form for selecting a file type to import

Description:

`get_import_fn()` is the main entry point. It creates a form that has a set of checkboxes for choosing a file type to import. Once selected the user then picks a file of that type to import.

The stucture is as follows:

```

/-----\
| filename                                     |
| _____ | _____ PCX                |
|                                     |
|                                     | _____ SIF |
|                                     |
| D | Q | _____ BIN                    | |
| | | |                                     |
| | | | _____ documentation              |
| | | | _____                            |
\-----/

```

Hardware: Compaq Despro 386/20

Compiler: Microsoft C version 5.1

Linker: `cl /c /AL %1.c`
`cl main binform /link met_bd1d mw_d1dmc`
`menuetml mnutlml /ST:4000`

Requires MetaWindows 3.6 and
Menuet 1.7E

Author: Karl Melder - Fall 1990

*/

```

/* ----- */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <GRconst.h>
#include <GRports.h>
#include <GRextrn.h>

#include "Menuet.h"
#include "project.h"

/* ----- */
/* Globally referenced variables..... */
/* ----- */

static fm_desc    *grList;          /* form for selecting a file */
static file_list  grFiles;          /* list of file names */

#define          PATH_LEN 40        /* The length of current fn path */

static char       path[PATH_LEN+1]; /* The current filename path */
static char       pathext[PATH_LEN+1]; /* Path plus filename extension */

/* ----- */
/* For List File */
/* ----- */

static char *lbls[] = { "Okay", "Path", "Quit" };
static int  keys[] = { 'O', 'P', 'Q' };
static int  stts[] = { 0, 0, 0 };

static char *noname[] = { "? ? ? ?" };

/* ----- */
/* NEW IMPORT.FRF 9/18/90 */
/* ----- */

char  *Imp_tag08_list[] =
{
    " Import will convert an image file in",
    " external format to ISIF format.",
    ""
};

```

```

mtag Imp_tag08 =
{
    Imp_tag08_list,
    3, 39
};

#define Imp_ob_CNT 9

ob_desc Imp_ob[] =
{
    /* 0 */
    { 0, obButton, 0, 0,
      attrLArr,
      IS_SELECTABLE|IS_INVERTIBLE|IS_CHECKABLE|IS_EXCLUSIVE,
      0,
      'M', 0, ".MSP ",
      NULL,
      { 328, 168, 347, 187 }, NULL },
    /* 1 */
    { 0, obButton, 0, 0,
      attrLArr,
      IS_SELECTABLE|IS_INVERTIBLE|IS_CHECKABLE|IS_EXCLUSIVE,
      0,
      'P', 0, ".PCx ",
      NULL,
      { 328, 198, 347, 217 }, NULL },
    /* 2 */
    { 0, obButton, 0, 0,
      attrLArr,
      IS_SELECTABLE|IS_INVERTIBLE|IS_CHECKABLE|IS_EXCLUSIVE,
      0,
      'B', 0, ".BIN ",
      NULL,
      { 328, 228, 347, 247 }, NULL },
    /* 3 */
    { 0, obButton, 0, 0,
      attrLArr,
      IS_SELECTABLE|IS_INVERTIBLE|IS_CHECKABLE|IS_EXCLUSIVE,
      0,
      'R', 0, ".RLE ",
      NULL,
      { 328, 258, 347, 277 }, NULL },
    /* 4 */
    { 0, obButton, 0, 0,
      attrLArr,
      IS_SELECTABLE|IS_INVERTIBLE|IS_CHECKABLE|IS_EXCLUSIVE,
      0,
      'G', 0, ".IMG ",
      NULL,
      { 328, 288, 347, 307 }, NULL },
    /* 5 */

```

```

{ 0, obButton, 0, 0,
  attrBut,
IS_SELECTABLE|IS_INVERTIBLE|IS_EXITABLE,
  0,
  0, 0, NULL,
  NULL,
  { 115, 179, 270, 211 }, NULL },
/* 6 */
{ 0, obButton, 0, 0,
  attrBut,
IS_SELECTABLE|IS_INVERTIBLE|IS_EXITABLE,
  0,
  'I', 0, "Import",
  NULL,
  { 114, 245, 181, 308 }, NULL },
/* 7 */
{ 0, obButton, 0, 0,
  attrBut,
IS_SELECTABLE|IS_INVERTIBLE|IS_EXITABLE,
  0,
  'Q', 0, "Quit",
  NULL,
  { 209, 245, 271, 308 }, NULL },
/* 8 */
{ 0, obMulLn, MLN_TAG, 0,
  attrArea,
IS_TAGGED,
  0,
  0, 0, "",
  (char *) &Imp_tag08,
  { 99, 335, 420, 390 }, NULL }
};

#define Imp_tx_CNT 2

txt_addn Imp_tx[] = /* Text to be put on form */
{
  { 300, 150, " Import Type? " },
  { 150, 150, " File Name? " }
};

#define Imp_ln_CNT 8

lin_addn Imp_ln[] = /* lines for drawing on form */
{
  { 297, 184, 297, 301 },
  { 301, 184, 301, 301 },
  { 121, 229, 261, 229 },
  { 111, 322, 410, 322 },
  { 147, 153, 241, 153 },
  { 303, 153, 409, 153 },

```

```

    { 302, 131, 409, 131 },
    { 146, 131, 244, 131 }
};

```

```

fm_desc Imp_fm =
{ 0, 4360, 0, attrDgBox ,
  IS_IDTGEN, -1,
  "",
  { 91, 120, 428, 398 },
  Imp_ob_CNT, Imp_ob,
  Imp_tx_CNT, Imp_tx,
  Imp_ln_CNT, Imp_ln };

```

```

char *msp[] =
{
  "MSP files are bitmap images produced by",
  "Microsoft paint packages",
  ""
};

```

```

char *pcx[] =
{
  "PCX/PCC files are bitmap images produced",
  "by Microsoft paint packages",
  ""
};

```

```

char *bin[] =
{
  "Binary files are bitmap images where",
  "1 pixel is encoded as 1 byte of data.",
  ""
};

```

```

char *rle[] =
{
  "Run Length Encoding compacts data, ",
  "where a sequence pixels with the same",
  "value are encoded as: [#ofpixels|value].",
  ""
};

```

```

char *img[] =
{
  "",
  "",
  ""
};

```

```

static

```

```
int    select_fn ();  
static  
int    mytask ();  
int    get_import_fn();
```

```

/* ----- */
/* Draw the display... */
/* ----- */

static int select_fn ( char fl_name[] )
{
    int    nslots,
           x, y;
    rect   r;
    int    ret,
           done,
           pick,
           itop,
           i;

    /* create list of image Files */

    Notify( -1, -1, "Please wait..." );
    ret = CreateFileList( &grFiles, pathext, 1 );
    NotifyOff();

    switch( ret )
    {
        case -1 :
            Beep();
            NotifyUser( -1, -1, "Error:", "Not enough memory" );
            return(-1);
        case 1 :
            Beep();
            NotifyUser(-1,-1,"Error:", "No Files found");
            return(-1);
        default: break;
    }

    /* create list form for files */

    nslots = 8;
    grList = CreateListForm( pathext, nslots, 3, 14, 6, 0 );
    SetObLabels( ListFormButtons(grList), lbls, 3 );
    SetObKeys( ListFormButtons(grList), keys, stts, 3 );

    OpenForm( grList, -1, -1 );

    /* Lets loop through until an event occurs */

    done = 0;
    while ( !done )
    {

```



```
        break;
    }
}

/* Lets Clean up after ourselves */
    CloseForm( grList );
    DestroyForm( grList );
    DestroyFileList( &grFiles );
}
```

```

/* ----- */
/* Task which is called from the control Loop to display online */
/* documentation */
/* ----- */

static int mytask ( ob_desc *ob )
{
    int      retv;
    app_msg  *aM;

    retv = USER_CONTINUE;
    aM = ReReadMsgQueue();

    /* Rewrite the multiple line field with documentation
    pertinent to the user. The field is changed each time
    the user selects a field to input */

    if( isMsPress(aM) ) /* Mouse down command */
    {
        switch ( ob->key )
        {
            case 'M': /* MSP file */
                SetMultObList( &Imp_ob[8], msp);
                DrawMultOb( &Imp_ob[8] );
                break;

            case 'P': /* PC? file */
                SetMultObList( &Imp_ob[8], pcx);
                DrawMultOb( &Imp_ob[8] );
                break;

            case 'B': /* BIN file */
                SetMultObList( &Imp_ob[8], bin);
                DrawMultOb( &Imp_ob[8] );
                break;

            case 'R': /* RLE file */
                SetMultObList( &Imp_ob[8], rle);
                DrawMultOb( &Imp_ob[8] );
                break;

            case 'G': /* IMG file */
                SetMultObList( &Imp_ob[8], img);
                DrawMultOb( &Imp_ob[8] );
                break;

        }
    }

    return (retv);
}

```

```

/*-----*/
/* Primary entry point. Get import image filename */
/*-----*/

int get_import_fn( char fl_name[], int *fl_type )
{
    int      ret,
            done, i;

    attr_blk  *ab;
    rect      r;

    /* Let make sure that look and feel of form is correct */

    ab = GetAttrBlk(attrDgBox);
    ab->bg_clr = ColorPalette[7];
    ab->fg_clr = ColorPalette[0];
    ab->thk = 1;
    ab->pat = 3;

    /* reset name place in button that get filename */

    strcpy(fl_name,"");          /* Lets clear it out */
    SetObLabels(&Imp_ob[5],noname,1);

    /* Let set up default of path to current directory */

    strcpy( path, CurrentDir() );
    strcpy( pathext, path );
    i = 0;
    while( path[i] != 0 ) i++;
    if ( path[i-1] != '\\' ) {
        strcat( path, "\\" );
        strcat( pathext, "\\" );
    }

    /* Lets put up the Import Dialog Box */

    HideCursor();
    OpenForm( &Imp_fm, -1, -1);

    /* Initialize the comment box */

    SetMultObList(&Imp_ob[8],Imp_tag08_list);
    DrawMultOb( &Imp_ob[8]);

```

```

ShowCursor();

/* Set up a link into processObList control loop to handle the
context dependent stuff */

SetUserKeyTask(mytask);

done = 0;
while( !done )
{
    ret = ProcessObList(Imp_ob,Imp_ob_CNT,0);

    switch(ret)
    {
        case 5 : /* Get file name */

            /* Lets initialize the path and add the extension */
            strcpy(pathext,path);          /* initialize path */

            if (Imp_ob[8].flags & IS_CHECKED) {
                strcat( pathext, ".MSP" );
                *fl_type = MSP;
            }

            else if (Imp_ob[1].flags & IS_CHECKED) {
                strcat( pathext, ".PC?" );
                *fl_type = PCC;
            }

            else if (Imp_ob[2].flags & IS_CHECKED) {
                strcat( pathext, ".BIN" );
                *fl_type = BIN;
            }

            else if (Imp_ob[3].flags & IS_CHECKED) {
                strcat( pathext, ".RLE" );
                *fl_type = RLE;
            }

            else if (Imp_ob[4].flags & IS_CHECKED) {
                strcat( pathext, ".IMG" );
                *fl_type = IMG;
            }

            /* Lets Now select a file name */

            ClearUserKeyTask();
            select_fn( fl_name );
            SetUserKeyTask(mytask);

```

```

/* Lets destroy the Old form and build a new one */

    HideCursor();
    CloseForm (&Imp_fm);
    OpenForm  ( &Imp_fm, -1, -1);
    ShowCursor();

    break;

case 6 : /* OK */

    /* Want to now process(import) the file selected */

    if ( strlen(fl_name) == 0 ) {
        NotifyUser(-1,-1,"OOPS", "A file must first be selected");
        done = 0;
    }
    else {
        done = 1;
    }

    break;

case 7 :  /* Quit */

    CloseForm( &Imp_fm );
    ClearUserKeyTask();
    return(-1);
    break;
}
}

ClearUserKeyTask();
CloseForm( &Imp_fm );
return(0);
}

```

```

/*-----*/
/*

```

Module Name: BINFORM.C

Purpose: To create a form for establishing parameters for loading a binary image file

Procedures:

```

get_binary_form() - create form and enter
                    control loop.
mytask()          - control loop task to change
                    documentation

```

Description: get_binary_form() builds a form which has the the following structure:

```

/-----\
| filename                                |
| _____ width _____ header     |
| _____ height _____            |
| |D| |Q|                               |
| |_____| |_____|                       |
| |_____ documentation _____|     |
\-----/

```

Once in the control loop, the users has the ability to select one of the fields (width, height, header) and change its values. As the user selects a field, the documentation changes to reflect the different requirements of that parameter.

These values are returned to the calling program.

Hardware: Compaq Despro 386/20

Compiler: Microsoft C version 5.1

```

Linker:          cl /c /AL %1.c
                  cl main binform /link met_bd1d mw_d1dmc
                  menuetml mnutlml /ST:4000

```

Requires MetaWindows 3.6 and
Menuet 1.7E

Author: Karl Melder - Fall 1990

```

                                                                    */
/* ----- */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <GRconst.h>
#include <GRports.h>
#include <GReutr.h>

#include "Menuet.h"
#include "project.h"

/* ----- */
/* Globally referenced variables... */
/* ----- */

static fm_desc    *grList;          /* form for selecting a file */
static file_list  grFiles;          /* list of file names */

#define          PATH_LEN 40        /* The length of current fn path */

static char       path[PATH_LEN+1]; /* The current filename path */
static char       pathext[PATH_LEN+1]; /* Path plus filename extension */

/* ----- */
/* For List File */
/* ----- */

static char *lbls[] = { "Okay", "Path", "Quit" };
static int  keys[] = { 'O', 'P', 'Q' };
static int  stts[] = { 0, 0, 0 };

static char *noname[] = { "? ? ? ?" };

/* ----- */
/* BIN BINARY.FORM 12/8/90 */
/* ----- */

char *BIN_tag00_list[] =
{
    "To import a binary image: set default width, ",
    "height, and header if incorrect. Then click on",

```

```

    "display button to open a window with the image."
};

mtag BIN_tag00 =
{
    BIN_tag00_list,
    3, 49
};

ftag BIN_tag01 =          /* Height */
{
    4, 0x0000, 0L,
    NULL,
    "\0\0\0\0"
};

ftag BIN_tag02 =
{
    4, 0x0000, 0L,
    NULL,
    "\0\0\0\0"
};

ftag BIN_tag05 =
{
    17, 0x0000, 0L,
    NULL,
    "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
};

ftag BIN_tag06 =
{
    5, 0x0000, 0L,
    NULL,
    "\0\0\0\0\0"
};

#define BIN_ob_CNT 7

ob_desc BIN_ob[] =
{
    /* 0 */                                /* User instruction */
    { 0, obMulLn, MLN_TAG, 0,
      attrArea,
      IS_SELECTABLE|IS_TAGGED,
      0,
      0, 0, "",
      (char *) &BIN_tag00,
      { 62, 317, 458, 372 }, NULL },
    /* 1 */                                /* Height of image in pixels */
    { 0, obField, FLD_TAG, 0,

```

```

    attrHArea,
    IS_SELECTABLE|IS_EDITABLE|IS_TAGGED|IS_INVERTIBLE,
    0,
    'H', 0, "Height ",
    (char *) &BIN_tag01,
    { 325, 248, 363, 268 }, NULL },
    /* 2 */ /* Width of the image in pixels */
{ 0, obField, FLD_TAG, 0,
    attrHArea,
    IS_SELECTABLE|IS_EDITABLE|IS_TAGGED|IS_INVERTIBLE,
    0,
    'W', 0, "Width ",
    (char *) &BIN_tag02,
    { 325, 219, 363, 239 }, NULL },
    /* 3 */ /* Display the image */
{ 0, obButton, 0, 0,
    attrBut,
    IS_SELECTABLE|IS_INVERTIBLE|IS_EXITABLE,
    0,
    'D', 0, "Display",
    NULL,
    { 74, 239, 144, 297 }, NULL },
    /* 4 */ /* Quit w/o getting image */
{ 0, obButton, 0, 0,
    attrBut,
    IS_SELECTABLE|IS_INVERTIBLE|IS_EXITABLE,
    0,
    'Q', 0, "Quit",
    NULL,
    { 158, 239, 228, 297 }, NULL },
    /* 5 */ /* Get FileName */
{ 0, obField, FLD_TAG, 0,
    attrArea,
    IS_TAGGED|IS_SELECTABLE,
    0,
    'F', 0, " File name ",
    (char *) &BIN_tag05,
    { 80, 193, 221, 212 }, NULL },
    /* 6 */ /* Header in bytes */
{ 0, obField, FLD_TAG, 0,
    attrArea,
    IS_SELECTABLE|IS_INVERTIBLE|IS_EDITABLE|IS_TAGGED,
    0,
    'E', 0, "Header",
    (char *) &BIN_tag06,
    { 390, 250, 436, 267 }, NULL }
};

```

```
#define BIN_tx_CNT 2
```

```
txt_addn BIN_tx[] =
```

```

{
    { 190, 163, "Import a Binary Image" },
    { 279, 203, " Image Dimensions " }
};

#define BIN_In_CNT 5

lin_addn BIN_In[] =
{
    { 278, 198, 254, 198 },
    { 454, 198, 424, 198 },
    { 254, 198, 254, 279 },
    { 454, 281, 454, 199 },
    { 254, 281, 454, 281 }
};

fm_desc BIN_fm =
{ 0, 4360, 0, attrForm1,
  IS_IDTGEN, -1,
  "",
  { 49, 135, 469, 380 },
  BIN_ob_CNT, BIN_ob,
  BIN_tx_CNT, BIN_tx,
  BIN_In_CNT, BIN_In };

char *height_txt[] =
{
    "Height contains the number of rows of pixels",
    "in the image.          ",
    ""
};

char *width_txt[] =
{
    "Width contains the number of columns of pixels",
    "in the image.          ",
    ""
};

char *header_txt[] =
{
    "Header is the number of bytes to skip at",
    "the beginning of the image file.      ",
    ""
};

char *filename_txt[] =
{

```

```
        "Filename is the name of the image file which ",  
        "will be displayed      ",  
        ""  
    };  
  
int    get_binary_form();  
static  
int    mytask();
```

```

/* ----- */
/* Primary entry point. Get import image filename */
/* ----- */

int get_binary_form( char fl_name[], int *xcols, int *yrows, long *header )
{
    int          ret,          /* return value */
              done,          /* done switch */
              i;              /* loop counter */

    attr_blk     *ab;          /* pointer to attribblock */
    rect         r;            /* rectangle */
    char         stuff[20];    /* buffer */

    dirRec       *dirinfo;     /* directory information */
    long         size_in_bytes; /* size in bytes */

    /* Initialize stuff */

    FlushMsgQueue(); /* We need to avoid having some message
                      in the Queue which needs to be read
                      in the control loop of the form */

    /* Set attributes of attrHArea to be inset just for this one */

    ab = GetAttrBlk( attrHArea );
    ab->bg_clr = LIGHTGREY;
    ab->fg_clr = BLACK;
    ab->thk     = 1;
    ab->bdr_style = 4; /* Inset border style */

    /* Check the size of the image file and determine it possible
       dimension. */

    if( FileQuery(fl_name,dirinfo,1) == 1)
        size_in_bytes = (long) dirinfo->fileSize;
    else NotifyUser(-1,-1,"ERR","file not found");

    if( size_in_bytes == 245760 ) {
        strcpy( BIN_tag01.fld, "480" ); /* height - yrows */
        strcpy( BIN_tag02.fld, "512" ); /* width  - xcols */
    }
    else if( size_in_bytes == 262144 ) {
        strcpy( BIN_tag01.fld, "512" ); /* height */
        strcpy( BIN_tag02.fld, "512" ); /* width */
    }
}

```

```

else if( size_in_bytes == 20000 ) {
    strcpy( BIN_tag01.fld, "100" );      /* height */
    strcpy( BIN_tag02.fld, "200" );      /* width */
}
else if( size_in_bytes == 61440 ) {
    strcpy( BIN_tag01.fld, "240" );      /* height */
    strcpy( BIN_tag02.fld, "256" );      /* width */
}
else {          /* Can't Guess at it */
    strcpy( BIN_tag01.fld, "" );          /* height */
    strcpy( BIN_tag02.fld, "" );          /* width */
}

/* Let instantiate the fields */

strcpy( BIN_tag05.fld, fl_name );        /* file name */
strcpy( BIN_tag06.fld, "0" );            /* header in bytes */

/* Lets put up the Import Dialog Box */

HideCursor();
OpenForm( &BIN_fm, -1, -1);

/* Initialize the comment box */

SetMultObList( &BIN_ob[0], BIN_tag00_list);
DrawMultOb( &BIN_ob[0] );
ShowCursor();

/* Set up a link into ProcessObList control loop to handle the
context dependent user help */

SetUserKeyTask(mytask);

/* Now, Let the user manipulate the interface */

done = 0;
while( !done )
{
    ret = ProcessObList(BIN_ob,BIN_ob_CNT,0);

    switch( ret )
    {
        case 3:          /* Display Image */

            done = 1;
            break;

        case 4:          /* Quit */

```

```
        ClearUserKeyTask();
        CloseForm( &BIN_fm );
        return(-1);
        break;
    }

}

/* save the row, width, and header data */

*yrows = atof(BIN_tag01.fld);
*xcols = atof(BIN_tag02.fld);

/* Ok, we are done, clean up and return an OK */

ClearUserKeyTask();
CloseForm( &BIN_fm );
return(0);
}
```

```

/* ----- */
/* task that is called from control loop to set documentation */
/* ----- */

static int mytask ( ob_desc *ob )
{
    int      retv;
    app_msg  *aM;

    retv = USER_CONTINUE;
    aM = ReReadMsgQueue();

    /* Determine in which field the user clicked, then set the multiple
       line field to new documentation */

    if( isMsPress(aM) ) /* Mouse down command */
    {
        switch ( ob->key )
        {
            case 'H': /* Height */
                SetMultObList( &BIN_ob[0], height_txt);
                DrawMultOb( &BIN_ob[0] );
                break;

            case 'W': /* Width */
                SetMultObList( &BIN_ob[0], width_txt);
                DrawMultOb( &BIN_ob[0] );
                break;

            case 'E': /* Header */
                SetMultObList( &BIN_ob[0], header_txt);
                DrawMultOb( &BIN_ob[0] );
                break;

            case 'F': /* Filename */
                SetMultObList( &BIN_ob[0], filename_txt);
                DrawMultOb( &BIN_ob[0] );
                break;

        }
    }

    return (retv);
}

```

```

/*-----*/
/*
Module Name: IMGLOAD3.C

Purpose:      To load an image into memory

Procedures:
    create_lcl_bmap(); - create a local memory bitmap
    destroy_lcl_bmap(); - destroy a local memory bitmap
    load_image_setup(); - prepare to create bitmap

Description:  load_image_setup() is the primary entry point.
               It first determines the size requirements for
               the incoming image, then creates a local memory
               bitmap to hold that image. It then loads the
               file into the local memory bitmap.

Hardware:     Compaq Despro 386/20

Compiler:     Microsoft C version 5.1

Linker:       cl /c /AL %1.c
               cl main binform /link met_bd1d mw_d1dmc
               menuetml mnutilml /ST:4000

               Requires MetaWindows 3.6 and
               Menuet 1.7E

Author:       Karl Melder - Fall 1990
*/-----*/

```

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <GRconst.h>
#include <GRports.h>
#include <GRextrn.h>

#include <menuet.h>
#include "project.h"
#include "gipsy.h"

#define FAR_ALLOC _fmalloc
#define FAR_FREE _ffree

int  create_lcl_bmap();
void destroy_lcl_bmap();
int  load_image_setup();

```

```

/* ----- */
/* Allocate memory for local memory bitmap */
/* ----- */

int create_lcl_bmap ( bMap, width, height, planes, bitpix, initflag )
bitmap *bMap;
int    width,          /* xcols in the image */
      height,         /* yrows in the image */
      planes,         /* number of planes in the image */
      bitpix,         /* number of bits per pixel */
      initflag;       /* initflag for initializing new memory */
{
    int i, n;
    unsigned char **raster_line;

    /* Set up all the parameters for a bitmap */

    bMap->devClass = -1;
    bMap->devType = 0;
    bMap->pixWidth = width;
    bMap->pixHeight= height;
    bMap->pixResX  = 100;
    bMap->pixResY  = 100;
    bMap->pixBits  = theBitmap->pixBits;
    bMap->pixPlanes= theBitmap->pixPlanes;
    bMap->rowBytes = ((width*bitpix) - 1)/8 + 1;

    SetSvMethod(svMEM);          /* set mem to alloc below 640K */

    for( n=0; n<planes; n++ )
    {
        raster_line = (unsigned char **)FAR_ALLOC(sizeof(char *)*(height));
        if ( raster_line == NULL )
            return( -1 );
        bMap->mapTable[n]= (map *) raster_line;
        for (i=0; i<height; i++, raster_line++)
        {
            *raster_line = (char *) FAR_ALLOC (bMap->rowBytes);
            if ( *raster_line == NULL )
                return( -1 );

            /* check initflag, if match init memory */
            if ( initflag == 0 )
                memset(*raster_line, 0x00, bMap->rowBytes);
            else if ( initflag == -1 )
                memset(*raster_line, 0xFF, bMap->rowBytes);
        }
    }
}

```

```
    SetSvMethod( svANY );      /* reset memory to use all forms */  
    return( 0 );  
}
```

```

/*-----*/
/* Destroy a local memory bitmap... */
/*-----*/

void destroy_lcl_bmap( bitmap *bMap )
{
    int i, n;
    unsigned char **raster_line;

    for( n=0; n<bMap->pixPlanes; n++ )
    {
        raster_line = (unsigned char **) bMap->mapTable[n];
        for (i=0; i<bMap->pixHeight; i++, raster_line++)
            FAR_FREE( *raster_line );
        FAR_FREE( bMap->mapTable[n] );
    }
}

```

```

/*-----*/
/* Load up the image into memory */
/*-----*/

int load_image_setup( fname, type, actwh, wninfo)
char  fname[];
int   type,
      actwh;
IMAGEWINDOW *wninfo ;
{
    int  ret,           /* return value */
         nbits,         /* number of bits per pixel */
         npple,         /* number of planes */
         nlins,         /* number of lines in image - yrows */
         ncols,        /* number of columns in image - xcols */
         nbnds,         /* number of bands in SIF image */
         nsbnds,        /* number of symbolic bands in SIF image */
         mode,          /* the mode of the SIF file */
         isif_type;     /* the internal sif type */

    rect      line;     /* rectangle of width 1 */
    image     *imgptr;   /* pointer to image */
    imageHeader *hdr;    /* pointer to image header */
    long      imbytes;   /* number of bytes to alloc for image */

    /* Hide the Cursor until loading is complete */

    HideCursor();

    /* Now, based on the type of image file, load file into image Map */

    switch ( type )
    {
        case PCC:
            /* Notify user it will take a while, and get
               size of file being loaded */

            Notify(-1,-1,"LOADING IMAGE - please wait...");
            QueryPCX(fname,  &wninfo->gbl_dx,
                       &wninfo->gbl_dy,
                       &wninfo->gbl_np,
                       &wninfo->gbl_nb );

            /* Create a large enough bitmap. If not enough
               memory, notify the user and close the window.*/

            ret = create_lcl_bmap( &wninfo->imagMap,
                                  wninfo->gbl_dx,

```

```

        wninfo->gbl_dy,
        wninfo->gbl_np,
        wninfo->gbl_nb, 0 );

if(ret == -1)
{
    NotifyOff();
    ShowCursor();
    NotifyUser(-1,-1,"System Error",
        "Cannot allocate enough memory to load image");
    SetActiveWn(0,1);
    return(-1);
}

/* Ok, now read in the file into the image bitmap,
   and then display it */

ReadPCX(fname, &wninfo->imagMap, 0, 0);
NotifyOff();

return(0);
break;

case SIF:
    /* Notify user it will take a while, and get
       size of file being loaded */

    Notify(-1,-1,"LOADING SIF IMAGE - please wait...");

    querySIF( fname, &nbits,
               &npple,
               &wninfo->gbl_dy,
               &wninfo->gbl_dx,
               &nbnds,
               &nsbnds,
               &mode,
               &isif_type);

    SetRect(&line,0,0,wninfo->gbl_dx,0);
    imbytes = ImageSize(&line);

    /* allocate memory */
    SetSvMethod(svMEM);          /* force under 640K */
    imgptr = (image *) FAR_ALLOC((unsigned short) imbytes);
    SetSvMethod(svANY);

    hdr = (imageHeader *) imgptr;
    ReadImage(&line,imgptr);
    FAR_FREE(imgptr);

```

```

/* set the header information for the window */
wninfo->gbl_np = hdr->imPlanes;
wninfo->gbl_nb = hdr->imBits;

/* create a local area bitmap to store image */
ret = create_lcl_bmap( &wninfo->imagMap,
                      wninfo->gbl_dx ,
                      wninfo->gbl_dy ,
                      hdr->imPlanes ,
                      hdr->imBits ,
                      0 );

if(ret == -1)
{
    NotifyOff();
    ShowCursor();
    NotifyUser(-1,-1,"System Error",
               "Cannot allocate enough memory to load image");
    SetActiveWn(0,1);
    return(-1);
}

/* Clean up and get out */
NotifyOff(); /* get rid of "please wait" */
return(0);
}
}

```

```

/ *-----* /
/ *

```

Module Name: WH2MANAGE.C

Purpose: To manage all the image windows.

Procedures:

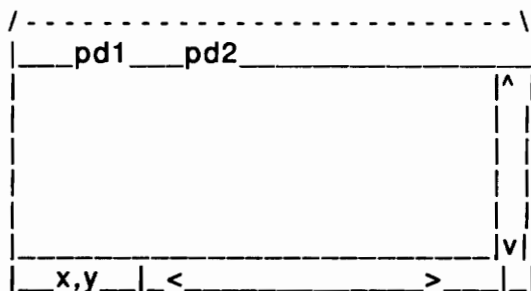
mkimagewin();	- make an image window
display_image();	- copy image to image window
draw_win_objects();	- draw objects (ie.scrollbars)
set_view_coordinates();	- set local & global coordinates
refresh_display_image();	- refresh the image
dowind();	- control loop for window
select_area();	- select a subarea of the image
window_select();	- user selects a window
Reposition();	- move the image in window
scroll_horz();	- scroll the image horizontally
scroll_vert();	- scroll the image vertically
do_vscroll();	- set vertical scroll direction
do_hscroll();	- set horizontal scroll direction

Description:

The three main functions in this module are:

- 1) mkimagewin();
- 2) display_image()
- 3) dowind()

1) mkimagewin() - makes a window that will display a bitmapped image. This procedure also draws the form object associated with each image window (i.e., the scrollbars, move button, and the [x,y] coordinate field.



2) display_image() - copies a region of the local bitmap to region of the screen bitmap. The regions are defined in the structure IMAGEWINDOW.

3) dowind() - this does the control loop functions

for the image window.

Hardware: Compaq Despro 386/20

Compiler: Microsoft C version 5.1

```

Linker:      cl /c /AL %1.c
             cl main binform /link met_bd1d mw_d1dmc
             menuetml mnutlml /ST:4000

```

**Requires MetaWindows 3.6 and
Menuet 1.7E**

Author: Karl Melder - Fall 1990

* /

/ * ----- *

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
#include <GRconst.h>
#include <GRports.h>
#include <GReutr.h>
```

```
#include <menuet.h>
#include "project.h"
```

```
#define TRUE 1
#define FALSE 0
```

```
/* main menus for image windows */
```

```
#define nWMAIN 8
```

[illegible]

```

fm_desc mainWMENU =
{
    fmHybrid, 0, 0, 0, 0, 0, NULL, 0, 0, 0, 0, nWMAIN, mainWITEMS,
    0, NULL, 0, NULL
};

#define nWSUB1 2
ob_desc sub1WITEMS[nWSUB1] =          /* Window pulldown */
{
    { 0,0,0,0,0,0,0,'R',0," Resize window ",NULL,0,0,0,0,NULL },
    { 0,0,0,0,0,0,0,'C',0," Close window ",NULL,0,0,0,0,NULL }
};
fm_desc sub1WMENU =
{
    0,0,0,0,0,0,0,NULL,0,0,0,0,nWSUB1,sub1WITEMS,0,NULL,0,NULL
};

#define nWSUB2 4                      /* Subimage pulldown */
ob_desc sub2WITEMS[nWSUB2] =
{
    { 0,0,0,0,0,0,0,'S',0," Select subimage ",NULL,0,0,0,0,NULL },
    { 0,0,0,0,0,0,0,'H',0," Histogram subimage ",NULL,0,0,0,0,NULL },
    { 0,0,0,0,0,0,0,'F',0," Fatbit subimage ",NULL,0,0,0,0,NULL },
    { 0,0,0,0,0,0,0,'V',0," saVe subimage ",NULL,0,0,0,0,NULL }
};
fm_desc sub2WMENU =
{
    0,0,0,0,0,0,0,NULL,0,0,0,0,nWSUB2,sub2WITEMS,0,NULL,0,NULL
};

#define FAR_ALLOC(n)    _fmalloc(n)
#define FAR_FREE(p)     _ffree(p)

#define WIDTH          260    /* minimum window width */
#define HEIGHT          150   /* minimum window height */

int      mkimagewin      ( );
void     display_image   ( );
static void draw_win_objects ( );
static void set_view_coordinates ( );
static void refresh_display_image ( );

int      dowind          ( );
static void select_area ( );
static int window_select ( );
static void Reposition   ( );

```

```
static void    scroll_horz  ( );  
static void    scroll_vert  ( );  
static void    do_vscroll   ( );  
static void    do_hscroll   ( );
```

```
extern char *fnt_bufr;
```

```
static IMAGEWINDOW *gbl_wninfo;  
static int gblA = 0;
```

```
/* Global for the control loop task */
```

```

/*-----*/
/*      Control loop task for dowind().  This procedure      */
/*      determines if mouse is within the window, if so it  */
/*      changes the cursor to a crosshair                    */
/*-----*/

static int mytask (int mx, int my)
{
    point mpos;          /* mouse position */
    int  retv;

    mpos.X = mx;
    mpos.Y = my;

    retv = USER_CONTINUE;

    if( PtInRect(&mpos, &gbl_wninfo->lcl_viewA) && gblA==0 )
    {
        CursorStyle(2);
        gblA = 2;
    }

    if( IPtInRect(&mpos, &gbl_wninfo->lcl_viewA)&& gblA==2)
    {
        CursorStyle(0);
        gblA = 0;
    }

    return(retv);
}

```

```

/*-----*/
/* process secondary window's main menu */
/*-----*/

int dowind( wh_to_activate, wninfo )
int wh_to_activate;
IMAGEWINDOW wninfo[];
{
    int i, /* loop counter */
        selection, /* the user selection */
        done, /* switch to break loop */
        cur_win; /* current window */
    int xloc,yloc; /* coordinates relative to image*/

    point mouse_loc, /* mouse position */
        *mouse_click; /* Record position of mouse */

    rect wrkR; /* The work rectangle */

    char *lbl[1];

    int sb_pick, /* button on the scroll bar */
        mainpick, /* pulldown pick */
        subpick; /* slot in pulldown */

    int ret; /* return value from a function */
    IMAGEWINDOW *r; /* pointer to image window */

    /* Now init some global variable for the control loop */

    gbl_wninfo = wninfo;

    /* Test to see if Window chosen by user is available, if not, the
       must make it first */

    mouse_click = &mouse_loc;

    /* Test whether image is obscured, then select passed window, */

    if ( WnObscured( wh_to_activate ) )
        ExposeWindow( wh_to_activate );
    else
        SetActiveWn( wh_to_activate, 1 );

    /* Set up loop to select objects on the window */

    SetUserEchoTask(mytask); /* Task for the control loop */

```

```

done = 0;
while( ! done )
{
    /* get a menu selection or record the position of the
       mouse click. */

    if( (cur_win = QryActiveWn()) == 0) break;

    /* Set the these global vars for the control loop task */

    gbl_wninfo = &wninfo[cur_win] ;
    selection = SelectOb( mainWITEMS, nWMAIN, 0 );

    /* break down return into 4 bit blocks. The selectOb()
       returns a value that is encoded such that the group
       of items is represented and its subcomponents */

    mainpick = selection & 0x000F;
    subpick = (selection >> 4) & 0x000F;

    mouse_click = MouseXY(1);      /* Save the mouseclick pos. */

    /* process the selection */
    switch( mainpick )
    {
        case 0 : /* Window: Resize, Close */
            if( subpick == 0 )
            {
                ResizeWindow();
                HideCursor();

                set_view_coordinates( cur_win, &wninfo[cur_win] );
                draw_win_objects(cur_win, &wninfo[cur_win] );
                refresh_display_image( &wninfo[cur_win] );
                /* Refresh image to delete the*/
                ShowCursor();
            }
            break;

        /* Lets clean it up befor leaving */

        done = 1;
        CloseWindow(cur_win,1);
        DestroyWindow(cur_win);
        destroy_lcl_bmap(&wninfo[cur_win].imagMap);
        ShowCursor();
        SetActiveWn( 0, 1 );
        ClearUserEchoTask();
        return(cur_win);
    }
}

```

```

        break;          /* This will be changed */

case 1 : /* Subimage: Select, Histogram, Fatbit */
    if (subpick == 0 )
        select_area( cur_win, &wninfo[cur_win] );
    if (subpick == 1 ) {
        SetActiveWn( 0, 0 );
        histogram();
        SetActiveWn( cur_win, 1);
    }
    if (subpick == 2 )
        NotifyUser(-1,-1,"Note","Not Implemented");
    break;

case 2 : /* Unused Pulldown menu in menubar */
    break;

case 3 : /* Nothing Button */
    break;

case 4 : /* Vertical Scroll Bar */

    wninfo[cur_win].selected = FALSE;
        /* reset for new selected region */
    sb_pick = WhichTagPick(&mainWITEMS[4]);
    do_vscroll(sb_pick, &wninfo[cur_win] );
    break;

case 5 : /* Horizontal Scroll Bar */

    wninfo[cur_win].selected = FALSE;
        /* reset for new selected region */
    sb_pick = WhichTagPick(&mainWITEMS[5]);
    do_hscroll(sb_pick, &wninfo[cur_win] );
    break;

case 6 : /* Move Button */

    MoveWindow();
    HideCursor();

    set_view_coordinates( cur_win, &wninfo[cur_win] );
    draw_win_objects(cur_win, &wninfo[cur_win] );
    ShowCursor();
    break;

case 7 :
    Beep();
    break;

```

default:

```

/* Check if users has clicked in another window.
   If another window was clicked in, and it was
   not one of the image windows (1 thru 8) then
   assume the user want to go to main window. */

if( PtlInWindow(mouse_click, cur_win) == False)
{
    ret=window_select( cur_win,mouse_click,wninfo);

    if( ret == 0)      /* user selected window 0 */
    {
        SetActiveWn(0,1);
        ClearUserEchoTask();
        return(0);
    }
}

if( PtlInRect(mouse_click,
               &wninfo[cur_win].lcl_viewA) == True)
{
    /* Calculate the position of the pixel that
       the user has mouse clicked on */

    r = &wninfo[cur_win];
    xloc = mouse_click->X - r->lcl_viewA.Xmin;
    yloc = mouse_click->Y - r->lcl_viewA.Ymin;
    xloc += r->imagSrc.Xmin;
    yloc += r->imagSrc.Ymin;

    /* put the current pointers x,y pos */
    sprintf(lbl[0],"%d[%d,%d]",
           GetPixel(mouse_click->X,mouse_click->Y),
           xloc,
           yloc);

    SetObLabels(&mainWITEMS[7],lbl,1);

    /* Draw window objects with [x,y] label */

    draw_win_objects(cur_win, &wninfo[cur_win] );

    /* Now clear out label for next time */

    strcpy(lbl[0],"");
    SetObLabels(&mainWITEMS[7],lbl,1);
    CursorStyle(2);

```

```
        }  
        break;  
    }  
}  
  
/* restore default window */  
  
    SetActiveWn( 0, 1 );  
    ClearUserEchoTask();  
    return(0);  
}
```

```

/* ----- */
/*    Check if user clicked in other windows    */
/* ----- */

static int window_select( cur_win, mouse_click, wninfo )
int cur_win;          /* index of active window */
point *mouse_click;   /* [x,y] location of mouse click */
IMAGEWINDOW wninfo[]; /* window information array */
{
    int i;             /* counter */

    for ( i=1; i<8; i++) /* loop thru all 8 windows */
    {
        if( PtInWindow(mouse_click,i) && (i!=cur_win))
        { /* if click is in a window and
            not current active window */

            /* Clean up stuff current window (i.e., del the selection
               box if one had been written. */

            if(wninfo[cur_win].selected == TRUE )
            {
                wninfo[cur_win].selected = FALSE;
                refresh_display_image( &wninfo[cur_win] );
            }

            /* expose the new window selected by the user */
            cur_win = i;
            if ( WnObscured(cur_win) )
                ExposeWindow(cur_win);
            else SetActiveWn(cur_win,1);

            /* Reset the window coordinates and redraw window objects*/
            set_view_coordinates( cur_win, &wninfo[cur_win] );
            draw_win_objects(cur_win, &wninfo[cur_win]);

            return(cur_win);
        }
    }

    /* It wasn't window 1 thru 8, so it must have been the
       the main window (i.e., window 0). */

    cur_win = 0;
    return( cur_win );
}

```

```

/* ----- */
/*      Select area on the Image      */
/* ----- */

static void select_area( cur_win, wninfo)
int    cur_win;
IMAGEWINDOW    *wninfo;
{
    point *mouse_click;

    /* See if user has previously selected an area in the window, if not,
       reset the selection box to a default size and location. */

    if( (wninfo->slctR.Xmax==0) && (wninfo->slctR.Ymax==0) )
    {
        SetRect(&wninfo->slctR, MouseX(),
                MouseY(),
                MouseX() + charW*3,
                MouseY() + charW*2 );
    }

    ResizeRect (&wninfo->slctR, &wninfo->viewA, charW*3, charW*2, 0);
    PositionRect (&wninfo->slctR, &wninfo->lcl_viewA);

    /* Let loop around until user clicks outside the frame.
       The user can move or resize the selection area. */

    while(PtInRect(MouseXY(0),&wninfo->lcl_viewA))
    {
        if (PtInRect(MouseXY(0),&wninfo->slctR) )
        {
            ResizeRect (&wninfo->slctR,&wninfo->viewA,charW*3,charW*2,0);
            PositionRect (&wninfo->slctR, &wninfo->lcl_viewA);
        }
        else
        {
            mouse_click = MouseXY(1);
            break;
        }
    }

    /* Check where final mouse click occurred, if inside viewing window,
       the user has selected this area. If the user has clicked
       outside the window, the user indicates that no region is to be
       selected */

    if( PtInRect(mouse_click, &wninfo->lcl_viewA) )

```

```
{
    wninfo->selected = TRUE;
    refresh_display_image( wninfo );
}
else
{
    wninfo->selected = FALSE;
    refresh_display_image( wninfo );
}
}
```

```

/* ----- */
/* Reposition to a new location... */
/* ----- */

static void Reposition( int dx, int dy, IMAGEWINDOW *wninfo )
{
    int cur_win;

    cur_win = QryActiveWn();

    OffsetRect( &wninfo->imagSrc, dx, dy );
    CopyBits( &wninfo->imagMap,
              theBitmap,
              &wninfo->imagSrc,
              &wninfo->imagDst,
              &wninfo->viewA, 0 );
}

```

```

/* ----- */
/* Scroll left or right... */
/* ----- */

static void scroll_horz( int dx, IMAGEWINDOW *wninfo )
{
    int lastx,
        cur_win;    /* current active window */
    rect src,       /* source rectangle */
        dst,       /* destination rectangle */
        mov,       /* rectangle to be moved on screen */
        r;         /* temporary rectangle */

    cur_win = QryActiveWn();

    /* Set the necessary rectangle coordinates that correspond to the image
       stored in memory, and actual image that exists w/in a window. */

    DupRect( &wninfo->imagDst, &dst );
    DupRect( &wninfo->imagSrc, &src );
    DupRect( &dst, &mov );

    /* Because the ScrollRect() work in Local mode, must readjust
       values from global mode */

    DupRect( QryWnRect(cur_win, WN_PHYS_RECT), &r );
    OffsetRect(&mov, -r.Xmin, -r.Ymin);

    lastx = wninfo->imagMap.pixWidth - 1;

    /* Now set up the amount of displacement based on dx parameter value */

    if ( dx > 0 )
    {
        mov.Xmin += dx;
        dst.Xmin = dst.Xmax - dx + 1;
        src.Xmin = src.Xmax - dx + 1;
    }
    else
    {
        mov.Xmax += dx;
        dst.Xmax = dst.Xmin - dx - 1;
        src.Xmax = src.Xmin - dx - 1;
    }

    /* Move the screen bitmap using the rectangle defined by mov, then fill
       in the screen bitmap with the amount displaced by ScrollRect */

```

```
while( (src.Xmin > 0) && (src.Xmax < lastx) )
{
    ScrollRect( &mov, -dx, 0 );
    CopyBits( &wninfo->imagMap,
              theBitmap,
              &src,
              &dst,
              &wninfo->viewA, 0 );
    OffsetRect( &src, dx, 0 );
    if ( src.Xmin < 0 ) break;
    if ( src.Xmax > lastx ) break;
    OffsetRect( &wninfo->imagSrc, dx, 0 );
    if ( MsgInQueue() ) break;
}
}
```

```

/* ----- */
/* Scroll up or down... */
/* ----- */

static void scroll_vert( int dy, IMAGEWINDOW *wninfo )
{
    int lasty,
        cur_win;    /* current window */

    rect src,        /* source rectangle */
        dst,        /* destination rectangle */
        mov,        /* rectangle to be moved on screen */
        r;          /* temporary rectangle */

    cur_win = QryActiveWn();

    /* Set the necessary rectangle coordinates that correspond to the image
       stored in memory, and actual image that exists w/in a window. */

    DupRect( &wninfo->imagDst, &dst );
    DupRect( &wninfo->imagSrc, &src );
    DupRect( &dst, &mov );          /* mov is used to scroll a section
                                     of bitmap image either up or down */

    /* Because the ScrollRect() work in Local mode, must readjust
       values from global mode */

    DupRect( QryWnRect(cur_win, WN_PHYS_RECT), &r);
    OffsetRect(&mov, -r.Xmin, -r.Ymin);

    lasty = wninfo->imagMap.pixHeight - 1;

    /* Now set up the amount of displacement based on dy parameter value */

    if ( dy > 0 )
    {
        mov.Ymin += dy;
        dst.Ymin = dst.Ymax - dy + 1;
        src.Ymin = src.Ymax - dy + 1;
    }
    else
    {
        mov.Ymax += dy;
        dst.Ymax = dst.Ymin - dy - 1;
        src.Ymax = src.Ymin - dy - 1;
    }

    /* Move the screen bitmap using the rectangle defined my mov, then fill
       in the screen bitmap with the amount displaced by ScrollRect */

```

```

while( (src.Ymin > 0) && (src.Ymax < lasty) )
{
    ScrollRect( &mov, 0, -dy );
    CopyBits( &wninfo->imagMap,
              theBitmap,
              &src,
              &dst,
              &wninfo->viewA, 0 );
    OffsetRect( &src, 0, dy );
    if ( src.Ymin < 0 ) break;
    if ( src.Ymax > lasty ) break;
    OffsetRect( &wninfo->imagSrc, 0, dy );
    if ( MsgInQueue() ) break;
}
}

```

```

/* ----- */
/* Process a vertical scroll button... */
/* ----- */

static void do_vscroll( which, wninfo )
int which;          /* which is the index # of the button selected in the
                    vertical scroll bar */
IMAGEWINDOW *wninfo; /* pointer to the window information */
{
    int    dy,          /* Displacement of y of cartesian plane */
    once;    /* Switch to indicate a scroll versus moving
              to some corner of the image */

    int    cur_win;    /* current window */

    /* Initialize the variables */

    dy = 0;
    once = 1;
    cur_win = QryActiveWn();

    /* Pick which button on the scroll bar was selected and perform the
       necessary action (i.e., scrollup, scrolldown, move_to_top,
       move_to_bottom). */

    switch( which )
    {
        case 2 : /* scroll image to top */
            dy = -wninfo->imagSrc.Ymin;
            break;

        case 3 : /* scroll image up one line at a time */
            dy = -1;
            once = 0;
            break;

        case 4 : /* scroll image down one line at a time */
            dy = 1;
            once = 0;
            break;

        case 5 : /* scroll image to bottom */
            dy = wninfo->imagMap.pixHeight -
                wninfo->imagSrc.Ymax - 1;
            break;
    }

    /* Now displace the image seen by the user by the specified amount. */

    if ( (once) && (dy != 0) )    Reposition( 0, dy, wninfo );
}

```

```
    else if ( dy != 0 )        scroll_vert( dy, wninfo );  
}
```

```

/* ----- */
/* Process a horizontal scroll button... */
/* ----- */

static void do_hscroll( which, wninfo )
int which;          /* which is the index # of the button selected in the
                    vertical scroll bar */
IMAGEWINDOW *wninfo;
{
    int          dx,          /* Displacement of x of cartesian plane */
              once;          /* Switch to indicate a scroll versus moving
                              to some corner of the image */

    int          cur_win;     /* current window */

    /* Initialize the variables */

    dx = 0;
    once = 1;
    cur_win = QryActiveWn();

    /* Pick which button on the scroll bar was selected and perform the
       necessary action (i.e., scrollup, scrolldown, move_to_top,
       move_to_bottom). */

    switch( which )
    {
        case 2 : /* scroll image to the left */
            dx = -wninfo->imagSrc.Xmin;
            break;

        case 3 : /* scroll image to the left dx lines at a time*/
            dx = -3;
            once = 0;
            break;

        case 4 : /* scroll image to the right dx lines at a time */
            dx = 3;
            once = 0;
            break;

        case 5 : /* scroll image to the right */
            dx = wninfo->imagMap.pixWidth -
                  wninfo->imagSrc.Xmax - 1;
            break;
    }

    /* Now displace the image seen by the user by the specified amount. */

    if ( (once) && (dx != 0) )      Reposition( dx, 0, wninfo );
    else if ( dx != 0 )             scroll_horz( dx, wninfo );
}

```

}

```

/* ----- */
/* Display the image stored in the bitmap onto the screen */
/* ----- */

void display_image( int cur_win, IMAGEWINDOW *wninfo )
{
    /* Set port rects and copy to screen. Remember, the CopyBits() function
       only works with global coordinates. This function also initializes
       the input image to upper left corner. See MetaWindow manual for more
       details on global versus local coordinates. */

    DupRect(&wninfo->viewA, &wninfo->imagSrc);
    OffsetRect( &wninfo->imagSrc,
                -wninfo->viewA.Xmin,
                -wninfo->viewA.Ymin);

    /* Check to see if image to be viewed is larger than the actual
       image. If so, reduce its size. */

    if(wninfo->imagSrc.Xmax >= wninfo->gbl_dx)
        wninfo->imagSrc.Xmax = wninfo->gbl_dx - 1;

    if(wninfo->imagSrc.Ymax >= wninfo->gbl_dy)
        wninfo->imagSrc.Ymax = wninfo->gbl_dy - 1;

    DupRect(&wninfo->imagSrc, &wninfo->imagDst);
    OffsetRect(&wninfo->imagDst, wninfo->viewA.Xmin, wninfo->viewA.Ymin);

    EraseRect(&wninfo->lcl_viewA);          /* Clear it out */
    CopyBits(&wninfo->imagMap, theBitmap, &wninfo->imagSrc,
              &wninfo->imagDst,
              &wninfo->viewA, 0);
}

```

```

static void refresh_display_image( IMAGEWINDOW *wninfo )
{
    int  x_origin, y_origin; /* Origin of image source rectangle */
    int  cur_win;           /* current window */

    cur_win = QryActiveWn();

    EraseRect(&wninfo->lcl_viewA);

    /* Save original coordinate of the upper left corner of the
       current coordinates of image Source, which is the view the user
       has established. */

    x_origin = wninfo->imagSrc.Xmin;
    y_origin = wninfo->imagSrc.Ymin;

    /* Copy and check that rectangle is no larger than existing image
       bit map. If so, resize it. */

    DupRect(&wninfo->viewA, &wninfo->imagSrc);
    OffsetRect(&wninfo->imagSrc,
               -wninfo->viewA.Xmin,
               -wninfo->viewA.Ymin);

    /* Check to see if image to be viewed is larger than the actual
       image. If so, reduce its size. */

    if(wninfo->imagSrc.Xmax >= wninfo->gbl_dx)
        wninfo->imagSrc.Xmax = wninfo->gbl_dx-1;

    if(wninfo->imagSrc.Ymax >= wninfo->gbl_dy)
        wninfo->imagSrc.Ymax = wninfo->gbl_dy-1;

    DupRect(&wninfo->imagSrc, &wninfo->imagDst);

    OffsetRect(&wninfo->imagDst,
               wninfo->viewA.Xmin,
               wninfo->viewA.Ymin);
    OffsetRect(&wninfo->imagSrc, x_origin, y_origin);

    CopyBits(&wninfo->imagMap,
             theBitmap,
             &wninfo->imagSrc,
             &wninfo->imagDst,
             &wninfo->viewA, 0);

    if ( wninfo->selected == TRUE )
    {
        PenColor(15);
        PenPattern(25); /* draw in grey */
        RasterOp(6);    /* draw Inverse Invert*/
    }
}

```

```
        DrawSingleBdr(&wninfo->slctR,1,0);  
        RasterOp(0);           /* Reset to 'replace' */  
        PenPattern(1);         /* Reset to black */  
        PenColor(0);  
    }  
}
```

```

/* ----- */
/* Set the global coordinates that will define the location of the      */
/* image on the screen.                                                */
/* ----- */

static void set_view_coordinates( int cur_win, IMAGEWINDOW *wninfo )
{
    int      x_glboffset,
             y_glboffset;      /* Global offset for doing CopyBit */
    rect     wrkR,             /* Window Work area */
            phsR;              /* Window Physical area */

    int      x_origin, y_origin; /* Origin of image source rectangle */

    /* Get the necessary information from the window, and then set the
       the GLOBAL image coordinates which equal to the work area minus
       the realstate needed for the vertical and horizontal scroll bars */

    DupRect( QryWnRect(cur_win, WN_WORK_RECT), &wrkR);
    DupRect( QryWnRect(cur_win, WN_PHYS_RECT), &phsR);
    x_glboffset = phsR.Xmin;
    y_glboffset = phsR.Ymin;

    SetRect(&wninfo->viewA, wrkR.Xmin + x_glboffset,
            wrkR.Ymin + y_glboffset,
            wrkR.Xmax - charW*2 + x_glboffset + 2,
            wrkR.Ymax - charW*2 + y_glboffset + 2 );

    /* Set the LOCAL coordinates for the image window */

    SetRect(&wninfo->lcl_viewA,
            wninfo->viewA.Xmin - x_glboffset,
            wninfo->viewA.Ymin - y_glboffset,
            wninfo->viewA.Xmax - x_glboffset,
            wninfo->viewA.Ymax - y_glboffset );

    /* Set the GLOBAL window coordinates for new window positions */

    x_origin = wninfo->imagSrc.Xmin;
    y_origin = wninfo->imagSrc.Ymin;

    DupRect(&wninfo->viewA, &wninfo->imagSrc);
    OffsetRect(&wninfo->imagSrc,
              -wninfo->viewA.Xmin,
              -wninfo->viewA.Ymin);

```

```
/* Check to see if image to be viewed is larger than the actual  
image. If so, reduce its size. */
```

```
    if(wninfo->imagSrc.Xmax >= wninfo->gbl_dx)  
        wninfo->imagSrc.Xmax = wninfo->gbl_dx - 1;
```

```
    if(wninfo->imagSrc.Ymax >= wninfo->gbl_dy)  
        wninfo->imagSrc.Ymax = wninfo->gbl_dy - 1;
```

```
    DupRect(&wninfo->imagSrc, &wninfo->imagDst);  
    OffsetRect(&wninfo->imagDst, wninfo->viewA.Xmin, wninfo->viewA.Ymin);  
    OffsetRect(&wninfo->imagSrc, x_origin, y_origin);
```

```
}
```

```

/* ----- */
/* File:   WHMAKER.C                               */
/* Desc:   program to make a window for an image   */
/* ----- */

int mkimagewin( filename, wninfo)
char  filename[];      /* The window name i.e., the image file */
IMAGEWINDOW wninfo[]; /* Pointer to window information structure */
{
    int  corner_x, corner_y, /* origin of the window */
        ret,                /* return value */
        new_window;         /* Index of new window */
    rect wrkR;               /* window work area */

    /* Hide the cursor so we don't mess up the screen */

        HideCursor();

    /* create window, and set the look and feel */

        SetWnBdrThk ( 2 );
        SetWnShwThk ( 0 );

        new_window = CreateWindow( filename, &mainWMENU, WIDTH, HEIGHT);

        if(new_window == -1) {
            NotifyUser(-1,-1,"ERROR","Window Table full");
            return(-1);
        }

        if(new_window == 0 ) {
            NotifyUser(-1,-1,"ERROR","New window returned 0");
            return(-1);
        }

        SetWnTtlColors ( new_window, 0, 7);
        SetWnColors    ( new_window, 0, 7 );

        SetWnMinSize(1, WIDTH, HEIGHT);      /* this only appears to work when
system                                           automatically determines sizes given
                                                objects does not work when resizing*/

    /* Initialize the menu/submenu tree */

        /* main menu item 1 */
        InitMenu( &sub1WMENU, ALIGN_VERT );

```

```

    AttachSubMenu( mainWITEMS, &sub1WMENU );

    /* main menu item 1 */
    InitMenu( &sub2WMENU, ALIGN_VERT );
    AttachSubMenu( mainWITEMS+1, &sub2WMENU );

    /* Determine some reasonable position to place the windows. Displace
       each window using width, height, and window number as parameters */

    DupRect( QryWnRect(0,WN_CLIENT_AREA), &wrkR );
    corner_x = charW + (charW * new_window) ;    /* offset image on x plane */
    corner_y = wrkR.Ymin + (charH * new_window);    /* offset image on y
plane */

    /* Now need to Open the Window, and activate it inorder to draw
       objects (i.e., scrollbar) within the window */

    OpenWindow ( new_window, corner_x, corner_y );
    SetActiveWn ( new_window, 1 );

    /* Draw the vertical scroll bar, the horizontal scroll bar, the move
       button, and the button to hold the mouse coordinates. */

    set_view_coordinates ( new_window, &wninfo[new_window] );
    draw_win_objects ( new_window, &wninfo[new_window] );
    ShowCursor();

    return ( new_window );
}

```

```

/* ----- */
/* Draw all the object (except menubar) with the newly created */
/* window, or a window that has been resized. */
/* ----- */

static void draw_win_objects(int cur_wh, IMAGEWINDOW *wninfo)
{
    int      x_glboffset,
             y_glboffset;      /* Global offset for doing CopyBit */
    rect wrkR;                 /* Window Work area */

    int      diffinx, diffiny,
             oldxpos, oldypos,
             newxpos, newypos;

    char *lbl[1];
    char msg[80];
    attr_blk *attrptr;

    /* Set the Window's Look and Feel */

    SetWnTtlColors(cur_wh,0,7);
    SetWnColors( cur_wh, 8, 7 );
    SetWnBdrThk(2);
    SetWnShwThk(0);

    /* The Horizontal Scrollbar needs to be reset for some reason */
    attrptr = GetAttrBlk(attrHArea);
    attrptr->bdr_style = 32;

    /* Build the area in which the image will reside */

    DupRect( QryWnRect(QryActiveWn(),WN_WORK_RECT),&wrkR);
    DrawSingleBdr(&wninfo->lcl_viewA,1,1);

    /* OK, lets build the vertical scroll bar */

    attrptr = GetAttrBlk(attrVArea);
    attrptr->thk = 1;

    InitSBar(&mainWITEMS[4],ALIGN_VERT, charW*2,
             wrkR.Ymax - wrkR.Ymin - charW*2+6);

    /* Now lets calculate the offset for the vertical scrollbar */

    newxpos = wrkR.Xmax - charW*2 + QryWnBdrThk() ;
    newypos = wrkR.Ymin + QryWnBdrThk() ;

```

```

oldxpos = mainWITEMS[4].area.Xmin;
oldypos = mainWITEMS[4].area.Ymin;

diffinx = newxpos - oldxpos + QryThk(attrVArea) + 2;
diffiny = newypos - oldypos + QryThk(attrVArea) - 4;

OffsetObList(&mainWITEMS[4],1,diffinx,diffiny);

DrawSBar(&mainWITEMS[4]);

/* OK, lets build the Horizontal scroll bar */

attrptr = GetAttrBlk(attrHArea);
attrptr->thk = 1;

InitSBar(&mainWITEMS[5],ALIGN_HORZ, charW*2, wrkR.Xmax - wrkR.Xmin
                                                charW*2 - 100);

/* Now lets calculate the offset for the scrollbar and draw it */

newxpos = wrkR.Xmin - QryWnBdrThk() ;
newypos = wrkR.Ymax - charW*2 + QryWnBdrThk();

oldxpos = mainWITEMS[5].area.Xmin;
oldypos = mainWITEMS[5].area.Ymin;

diffinx = newxpos - oldxpos + QryThk(attrHArea) + 106;
diffiny = newypos - oldypos + QryThk(attrHArea) + 2;

OffsetObList(&mainWITEMS[5],1,diffinx,diffiny);
DrawSBar(&mainWITEMS[5]);

/* Now put a button on the corner that will used to MOVE the window */

InitObFields(&mainWITEMS[6],1,-1,-1,attrBut,-1,-1);
InitObFlags(&mainWITEMS[6],1,IS_SELECTABLE|IS_INVERTIBLE);
attrptr = GetAttrBlk(attrBut);
attrptr->thk = 1;

SetRect(&mainWITEMS[6].area,wrkR.Xmax - charW*2 +
QryThk(attrHArea)*5,
                                wrkR.Ymax - charW*2 + QryThk(attrVArea)*5,
                                wrkR.Xmax,
                                wrkR.Ymax );

DrawOb(&mainWITEMS[6]);

```

```

/* Now put button on left of v.scroll bar, for displaying x,y coordinates */

InitObFields(&mainWITEMS[7],1,-1,-1,attrBut,-1,-1);
InitObFlags(&mainWITEMS[7],1,IS_PASSIVE);
attrptr = GetAttrBlk(attrBut);
attrptr->thk = 1;
SetRect(&mainWITEMS[7].area,wrkR.Xmin,
        wrkR.Ymax - charW*2 + QryThk(attrVArea)*5,
        wrkR.Xmin + 100,
        wrkR.Ymax - 1);

SetCurFont( (fontRec *) fnt_bufr ); /* set to small font */
DrawOb(&mainWITEMS[7]);
SetMnFont(); /* reset to main font */
}

```

```

/*-----*/
/*

```

Module Name: SELECTFLC

Purpose: To have the user select a file from the a list in the directory.

Procedures:

```

do_files( )      - does the control loop for form
get_file_name( ) - main function

```

Description: get file name is main function which opens a form that allows the user to select a filename. Do_files() handles the control loop.

Hardware: Compaq Despro 386/20

Compiler: Microsoft C version 5.1

Linker: cl /c /AL %1.c
 cl main binform /link met_bd1d mw_d1dmc
 menuetml mnutlml /ST:4000

Requires MetaWindows 3.6 and
 Menuet 1.7E

Author: Karl Melder - Fall 1990

```

/*-----*/
*/

```

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

```

```

#include <GRconst.h>
#include <GRports.h>
#include <GReptrn.h>

```

```

#include "menuet.h"
#include "project.h"

```

```

#define TRUE 1
#define BLANK ""

```

```

#define PATH_LEN 40          /* buffer to hold a path/file-spec */
char path[PATH_LEN+1];

```

```

#define nBUTT 3              /* button count */
#define bWID 7               /* button width in characters */

```

```
static
void  do_files( );          /* does the control loop for form */
int   get_file_name( );    /* main function */

file_list fList;           /* file list structure */

/* button list for list form */
char *but_text[nBUTT] =
{
    "OK",
    "Path",
    "Cancel"
};
```

```

/*****
/ *   Get the file name.
/ *****/

int get_file_name( char filename[], int file_type )
{
    int i,          /* loop counter */
        done;       /* exit switch */

    /* Initialize path buffer to current directory */
    strcpy( path, CurrentDir() );

    i = 0;
    while( path[i] != 0 ) i++;
    if ( path[i-1] != '\\' )
        strcat( path, "\\" );
    strcat( path, "**." );

    switch( file_type ) /* choose the extension type */
    {
        case SIF:
            strcat( path, "SIF" );
            break;
        case PCC:
            /* include PCX */
            strcat( path, "PC?" );
            break;
    }

    do_files(filename); /* go into control loop */
    if( strlen(filename) == 0 )
        return(-1);      /* return filename not chosen */
    else return(0);      /* return filename chosen */
}

```

```

/*****
/ *   Get the files, then enter the control loop
/ *****/

void do_files( filename )
char filename[];
{
    int i,           /* loop counter */
        retv,       /* return value */
        pick,       /* users pick */
        top;        /* top of the list */

    char msg[80];    /* message */
    fm_desc *lf;     /* pointer to form */
    ob_desc *ob;     /* pointer to an object */

    /* let user enter a path spec */
    while (TRUE)
    {
        /* create a list of matching files */
        retv = CreateFileList( &fList, path, 1 );

        /* check for possible error returns */
        switch( retv )
        {
            case 1 : /* no files found */
                sprintf( msg, "No files found that match\n'%s'", path );
                NotifyUser( -1, -1, "Note:", msg );
                return;
            case -1 : /* out of memory */
                NotifyUser( -1, -1, "Error:", "Out of memory\nallocating file list" );
                return;
        }

        /* create list form */
        lf = CreateListForm( path, 7, nBUTT, 14, bWID, 0 );

        /* add our button labels and keys... */
        /* note: by default, buttons are exitable */
        ob = ListFormButtons( lf );
        for( i=0; i<nBUTT; i++ )
        {
            ob->label = but_text[i];
            ob->key   = *but_text[i];
            ob++;
        }

        /* and open the form */
        OpenForm( lf, -1, -1 );
    }
}

```

```

/* select one of the files */
pick = top = 0;
retv = ProcessListForm( lf, fList.fnames, fList.nfiles,
                        &top, &pick );

if(retv == 9)          /* User OK */
{
    strcpy(filename,fList.fnames[pick]);
    break;
}

if(retv == 10)         /* User want change dir path*/
    GetString(-1,-1,
              "Enter Path/File Specifications:",
              path,PATH_LEN);

if(retv == 11)        /* User canceled out */
{
    strcpy(filename,""); /* delete out whats there */
    break;
}

/* get rid of list form */
CloseForm( lf );
DestroyForm( lf );

/* and get rid of file list */
DestroyFileList( &fList );
}

/* Let clean it up */

CloseForm( lf );
DestroyForm( lf );
DestroyFileList( &fList );
}

/*****
/*  Get a SIF Image File and Load Into Internal SIF Structure (ISIFS) */
*****/
/*
Module:          RSIF7.C

Function:        load_sif_image()

Input Params:
    ISIFS *asifsimage - pointer to Internal SIF structure
    char *filename - pointer to the name of SIF file
    int *filetype - either a VAX or a PC type SIF

```

Calls:

```

load_initial_record()
load_ident_record()
load_history_records()
decide_type()
char_convert()
short_convert()
float_convert()

```

Purpose: This procedure reads a SIF image file and loads it into Internal SIF Structure (ISIFS).

Description:

A SIF image file is broken down into four major components:

```

Initial Record;
Identification Record;
History Record; and
Image Data.

```

The Initial Record contains the record length in bytes. This info is used to determine the beginning of the Identification Record. The Identification Record specifies the number of History Records and the format of the Image Data. All records within a SIF file are of equal length.

The Image Data can be encoded in various ways. The information required to determine this encoding includes:

```

nbits - number of bits per picture element
mode - data mode
nbnds - number of bands

```

GIPSY compacts integer data based on the range of image data values (i.e., idmin, idmax). The GIPSY system allocates X bits per picture element depending on the number of bits required to represent the full range of possible values. This integer data can also be formatted in 2's complement for the purpose of representing negative numbers. The mode indicates whether or not 2's complement is implemented.

For most SIF files, one record represents one line of image data. Therefore, the number of records allocated for data is also the number of rows in a given image. For multiband images, several records can represent one line of image data. For example, an RGB image has three bands, each band representing one of the primary colors. GIPSY will store them as three sequential records. The total number of records (i.e., rows) stored for the image will therefore be the number of rows in the image times the number of bands.

The ISIFS structure tries to maintain both the structure and the data found in the SIF files. The ISIFS has an Initial Record Structure, an Identification Record Structure, a pointer to the history records, and a pointer to the data (i.e., picture elements). At present, even the first four bytes of each record, which is part of the VAX record structure, is stored. However, some changes are performed. First, the Initial Record and the Identification Records does not store the padding. The History Records are simply recorded as char data. A good deal of conversions are performed on the actual image data. The image data is converted into one of three formats:

- 1) unsigned char (1-byte)
- 2) signed short (2-bytes)
- 3) float (4-bytes)

Some of the conversions into float do result with a loss of precision. The user must be informed of this loss of precision when it occurs. Some of the conversions are optimized given the original SIF data structure. For instance, if the original data is in 8-bit unsigned integer format, then a simple memory copy is sufficient to load the data into memory.

Currently, this program will not do SIF files with multiple rows per record, or SIF files with data formats of type float. The problem with SIF files of type float is that VAX does not use a standard ASCII encoding scheme.

Method:

1) Load the first several words of the first record of the SIF file. These first words contain the byte length of the records. This length does not include the first four bytes (the VAX record header) found at the beginning of each record of a VAX generated SIF file. The information found in this first record include:

idnrcsfl, - Number of Records in the SIF file
idnbtsrc, - Number of Bits in a record

2) Since now the length of each record is known, we can then position the file pointer to the beginning of each record in the file. The next record we need is the identification record, which is always the second record in the file. The major information on the structure of the file, and the format of the data is contained in the identification record. This includes:

idusr1, - Unused
idusr2, - Unused
idsys1, - Unused and reserved
idsys2, - Unused and reserved
idnbits, - Number of bits/pixel
idnppl, - Number of pixels/line

idnlns,	- Number of lines in an image
idhrxs,	- Relative Horiz pixel dimension
idvrxs,	- Relative Vert pixel dimension
idnscrs,	- Number of descriptor records in file
idnql,	- Number of quantized grey levels
idnwds,	- Number of words per record
idncols,	- Number of columns in each subimage
idnrows,	- Number of rows in each subimage
idmin,	- Minimum grey tone over all bands
idmax,	- Maximum grey tone over all bands
idnbnds,	- Total number of bands in an image
idnsbnds,	- Number of symbolic (map) bands
idmode,	- Data mode indicator
idver;	- Version number of sif format

3) The pointer is then positioned at the third record, which typically begins the history records. The number of history records is contained in idscrs (referred to as descriptor records). Memory is allocated and these records are stored as text.

4) It must then be decided which variable type is needed to represent the incoming data. This decision is based on the number of bits are used to represent the picture values (idnbits) and the mode of the data (idmode).

5) Based on the decision, the data is converted one line at a time. Given some of the initial data formats of the SIF file, the need for a conversion is negated, and a simple copy of data byte for byte is sufficient. In the worst case, data must be read bit-by-bit and then converted. This is optimized a bit by loading a record of data into memory, and then performing the conversion.

6) The only thing that remains to do is to change some of the header information, in what can now be referred to as the ISIFS (Internal SIF structure), to reflect the conversions. This includes the number of bits needed to represent one picture element, number of bits in a record, the mode of the data, and the number of words in each record.

NOTE: There are some distinct differences between the ISIFS and the SIF structure as produced by the VAX. As stated earlier, the GIPSY image processing system on the VAX produces an image that is in SIF format. As such, the file is segmented into records, with the VMS OS placing a 4 byte header at the beginning of each record. While running on the VAX, the structure is hidden to the user. However, when transferred to the personal computer, this file exists as a stream - including the 4 bytes put there by VMS OS. The result is several considerations. First, the transfer of an image back to VAX will involve having a program on VMS to read the file in stream mode, then writing it out in record mode. Secondly, software will

have to distinguish between SIF files in VAX format from SIF files in PC (i.e., w/o 4 byte header) format. Finally, the most difficult issue is the different floating point formats between machines. Some of the compilers use the IEEE standard. The VAX C compiler does not. This makes floating point data, as produced by the VAX, unreadable by programs not generated on the VAX platform.

Hardware: Compaq Deskpro 386/20
 IBM AT

Compiler Metaware High C compiler - version 1.6
 Microsoft C compiler - version 5.1

Author: Karl Melder - August, 1990

Future Updates:

- 1) Take into account symbolic bands
- 2) The issue of VAX floating point vs IEEE floating point.
- 3) Remove all I/O functions, and simply returning error codes.

*****/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
```

```
#include <GRconst.h>
#include <GRports.h>
#include <GReptrn.h>
```

```
#include <menuet.h>
#include "gipsy.h"
```

```
#define VAXOFFSET                4                /* Header size for each VAX record */
#define XTRA_INT        sizeof(int)           /* Primarily for xtra space for alloc */
```

```
#define TYPE1            unsigned char *
#define TYPE2            signed short *
#define TYPE3            float *
```

```
#define        isbetw(x,low,high) ( (x >= low) && (x<=high) ? 1 : 0)
```

```
ISIFS        *simage;                /* Pointer to an internal SIF image */
FILE        *fptr;                /* File pointer */
```

```
static void    load_initial_record();
```

```
static void    load_ident_record();
static void    load_history_records();
static int     decide_type();
static void    *char_convert();
static void    *short_convert();
static void    *float_convert();
static void    convert_all();
```

```

void load_sif_image(filename,asifimage)
char  *filename;          /* Pointer to SIF file */
ISIFS *asifimage;         /* Pointer to Internal SIF Struct */
{
    long  bytestoalloc;    /* How many bytes to alloc for 1 image*/
    int   type,            /* The type of SIF file {VAX,PC} */
          recl_b,          /* Record length in bytes for SIF */
          newrecbsize,     /* Record size in bytes for ISIFS */
          newrecwsize;     /* Record size in words for ISIFS */

    /* Assign the global simage pointer to the address of the ISIFS */

    simage = asifimage;

    /* Open up the image file under read only and binary mode */

    if ((fptr = fopen(filename,"rb")) == NULL)
    {
        NotifyUser(-1,-1,"DEBUG","Not able to open the file.\n");
        exit(-1);
    }

    /* Load the initial record, Calculate length of the record, load the
       identification record, and then the history records. */

    load_initial_record();
    recl_b = ((int)simage->inrc.idnbtsrc / 8) + VAXOFFSET;
    load_ident_record(recl_b);
    load_history_records(recl_b);

    /* Lets now decide what the internalsif looks like it can be of: type = 1
       (unsigned 8 bit); type = 2 (signed 16 bit), or type = 4 (32 bit float).
       Note that the assignment of type is equivalent to the number of
       bytes required to represent the data type. */

    if( (type = decide_type( (int)simage->idrc.idmode,
                           (int)simage->idrc.idnbits) ) == 0)
        NotifyUser(-1,-1,"DEBUG","ERROR - cannot assign a type.\n");

    /* Lets calculate the size in bytes that the new record size needs to
       to be. Then determine how many words are required, using the modulo
       to determine if an extra word is needed. */

    newrecbsize = (int) simage->idrc.idnppl * type;

    if (newrecbsize % 4 == 0) /* mod to check if remainder exists */

```

```

        newrecwsize = newrecbsize/4;
    else
        newrecwsize = newrecbsize/4 + 1;

    /* Size of record cannot be smaller than the size of Ident_Structure,
       so if smaller use old values. Also recalculate newrecbsize */

    if(newrecwsize < simage->idrc.idnwds)
        newrecwsize = simage->idrc.idnwds;
    newrecbsize = (newrecwsize * 4) + VAXOFFSET;

    /* Now lets allocate memory based on type we have choosen for ISIFS
       representation. The amount of memory to allocate is the new
       record size times number of bands plus an extra few bytes at
       the end for safety */

    bytestoalloc = (long)newrecbsize * simage->idrc.idnbnds + XTRA_INT;
    if((simage->data_ptr = (void *) calloc(bytestoalloc,1)) == NULL)
        NotifyUser(-1,-1,"DEBUG","ERROR - Insufficient memory available to alloc
        for image.\n");

    /* Ok, lets load the data one line at a time, saving tptr as we go. The
       updated pointer is returned by the convert function, which converts the
       data from the SIF file into ISIFS format, and copies into the memory
       allocated for data. */

    convert_all(fp_ptr,type,rec_l_b,newrecbsize,newrecwsize);

    fclose(fp_ptr);
}

```

```

void load_initial_record(void)
{
    int retvalue;

    /* ***** */
    /* PURPOSE: To read in the beginning (first several words) of a      * /
    /* SIF image file into the internal SIF structure. The information    * /
    /* includes the length of the records and the number of records in    * /
    /* the SIF image file.                                              * /
    /* ***** */

    /* Set file ptr to 0, and copy byte-by-byte to structure InitialSifRec
    in ISIFS. */

    retvalue = fseek(fp, (long) 0, SEEK_SET);
    if (retvalue != 0)
        NotifyUser(-1, -1, "DEBUG", "Fseek failed to beginning.\n");

    /* Read initial record (See GIPSY handbook) into Internal SIF Structure
    (ISIFS) exactly the number of bytes in the InitialSifRec. */

    fread(&(simage->inrc), sizeof(char), sizeof(InitialSifRec), fp);
}

```

```

void load_ident_record(recl_b)
int recl_b;          /* The length of a record in bytes */
{
    int retvalue;      /* Return Value from a function call */

    /* ***** */
    /* PURPOSE: To read in the identification record of a SIF image      * /
    /* file into the internal SIF structure.                             */
    /* ***** */

    /* Move file pointer to second record in the SIF image file. This
       position is determined from information supplied in the initial record.*/

    retvalue = fseek(fp, (long)recl_b, SEEK_SET);
    if (retvalue != 0)
        NotifyUser(-1, -1, "DEBUG", "Fseek failed to second record.\n");

    /* Read the identification record (See GIPSY handbook) into Internal
       SIF Structure (ISIFS).*/

    fread(&(simage->idrc), sizeof(char), sizeof(IdentSifRec), fp);
}

```

```

void load_history_records(recl_b)
int  recl_b;          /* The length of a record in bytes */
{
    int retvalue;      /* Return Value from a function */
    long bytestoalloc, /* Bytes to allocate in malloc */
        bytestoread;   /* Bytes to Read for file */

    /******
    /* Purpose: To allocate memory and load in the history records          */
    /* from the SIF image file. How many of these history files that      */
    /* exist is found in the identification record of the SIF image.      */
    /******

    /* Allocate enough memory to hold entire set of history records.*/

    bytestoalloc = (long)recl_b * simage->idrc.idnscrs + XTRA_INT;
    if((simage->history_ptr = malloc(bytestoalloc)) == NULL)
        NotifyUser(-1,-1,"DEBUG","ERROR - could not alloc memory for history
file\n");

    /* Move pointer to the beginning to the history records (third record
    in SIF image file). */

    retvalue = fseek(fp, (long)recl_b * 2, SEEK_SET);
    if (retvalue != 0)
        NotifyUser(-1,-1,"DEBUG","ERROR - could not position file pointer.\n");
    bytestoread = (long)recl_b * simage->idrc.idnscrs;
    fread(simage->history_ptr, sizeof(char), bytestoread, fp);
}

```

```

int decide_type( int mode, int nbits )
{
    /* ***** */
    /* Purpose: Determines the encoding of the image data, and returns */
    /* an appropriate variable type for internal SIF representation. */
    /* ***** */

    /* also check that number of band = 1 */
    if ( (mode == 0) && isbetw(nbits,1,8) )
        return(1); /* Unsigned Char */

    if ( mode == 0 && isbetw(nbits,9,16) )
        return(4); /* float */

    if ( mode == 1 && isbetw(nbits,1,16) )
        return(2); /* Signed Short */

    if ( mode == 1 && isbetw(nbits,17,32) )
        return(4); /* float */

    if ( mode == 0 && isbetw(nbits,17,32) )
        return(4); /* Float */

    if (mode == 2)
        return(4); /* Float */

    /* If nothing is satisfied, return 0 */
    return(0);
}

```

```

static void convert_all(fp_ptr,type,recl_b,newrecbsize,newrecwsize)
FILE *fp_ptr;
int type;
int recl_b;
int newrecbsize;
int newrecwsize;
{
    int i;                /* Loop counter */
    void *Inbuf;          /* Pointer to 1 record length buffer */

    /* Allocate memory for one line of incoming image data. */

    if((Inbuf = (void *) malloc(recl_b + XTRA_INT)) == NULL)
        NotifyUser(-1,-1,"DEBUG","ERROR - could not alloc memory of Inbuf.\n");

    /* Now, after deciding what kind of conversion will be performed,
       read a line of data (one record) from the SIF file and convert
       the data. Then load this converted data into memory. This process
       must performed the number of rows (i.e., data records) the the image
       has. For multiband images, however, must convert the number of rows
       times the number of bands w/in the image. */

    switch (type)
    {
        case 1:

            /* Now lets loop through record for record and convert data */

            for(i=0; i<simage->idrc.idnlns * simage->idrc.idnbnds; i++)
            {
                fread(Inbuf,sizeof(char),(long)recl_b,fp_ptr);
                char_convert( (TYPE1) Inbuf,
                             (TYPE1) simage->data_ptr,
                             recl_b,
                             newrecbsize);
            }

            /* Set IFSIFS structure for new data structure values */
            simage->idrc.idnbits = 8;
            simage->idrc.idmode = 0;
            simage->idrc.idnwds = (long) newrecwsize;
            simage->inrc.idnrcsfl = (long) ((newrecbsize - 1) * 8);
            break;

        case 2:

            for(i=0; i<simage->idrc.idnlns * simage->idrc.idnbnds; i++)

```

```

{
    fread(Inbuf,sizeof(char),(long)recl_b,fptr);
    short_convert( (TYPE2) Inbuf,
                  (TYPE2) simage->data_ptr,
                  recl_b,
                  newrecbsize);
}

/* Set IFSIFS structure for new data structure values */
simage->idrc.idnbits = 16;
simage->idrc.idmode = 1;
simage->idrc.idnwds = (long) newrecwsize;
simage->inrc.idnrcsfl = (long) ((newrecbsize - 1) * 8);

break;

case 4:
    NotifyUser(-1,-1,"DEBUG","doing case 4\n");

    for(i=0; i<simage->idrc.idnlns * simage->idrc.idnbnds; i++)
    {
        fread(Inbuf,sizeof(char),(long)recl_b,fptr);
        float_convert( (TYPE3) Inbuf,
                      (TYPE3) simage->data_ptr,
                      recl_b,
                      newrecbsize);
    }

    /* Set IFSIFS structure for new data structure values */
    simage->idrc.idnbits = 32;
    simage->idrc.idmode = 2;
    simage->idrc.idnwds = (long) newrecwsize;
    simage->inrc.idnrcsfl = (long) ((newrecbsize - 1) * 8);

    break;

case 5:
    NotifyUser(-1,-1,"DEBUG","Not yet available\n");
    break;
}
free((void *)Inbuf);
}

```

```
double powerof( double x, double y )
{
    int i;
    double temp;

    /* A quick function to determin x to the power of y */

    temp = 1.0;
    for( i=1; i <= y; ++i )
    {
        temp = temp * x;
    }
    return(temp);
}
```

```
int scale( int number, double sc_from, double sc_to )
{
    double new_number, dbl_from, dbl_to;

    sc_from = (double) (powerof( 2, sc_from) - 1);
    sc_to  = (double) (powerof( 2, sc_to ) - 1);

    new_number = (double) (sc_to / sc_from) * number;
    new_number += 0.5;          /* add .5 for rounding off */

    return( (int) new_number);  /* Send back truncated # */
}
```

```

void *char_convert(Inbuf,destptr,recl_b,newrecbsize)
unsigned char *Inbuf, *destptr;
int recl_b, newrecbsize;
{
    unsigned char bit[8];          /* Used to set bits in a byte */
    unsigned char bitbuffer;       /* A buffer to build 1 byte of data */
    int bitcntr;                  /* A bit counter */
    int i,j, destcntr;            /* Incrementers */
    unsigned int powit;
    char msg[80];

    /* Load bit - note, wanted to pow() but double to uns char is nasty. Essentially, we load do a: bit[8] = {1,2,4,8,16,32,64,128}; */

    powit = 1;
    for (i=0; i<8; i++) {
        bit[i] = (unsigned char) powit;
        powit *= 2;
    }

    /******
    /*      Super Fast image loading                                     */
    /*      if unsigned 8-bit, copy memory byte for byte               */
    /******

    if (simage->idrc.idmode == 0 && simage->idrc.idnbits == 8)
    {
        memcpy(destptr,Inbuf, recl_b);
        return( (void *) (destptr + recl_b)); /* Return updated pnter */
    }

    /******
    /*      Slow image loader, 1-7 bits                                 */
    /*      if (nbits >= 1-bit and <= 7) && mode is unsigned integer    */
    /******

    if((simage->idrc.idmode==0) && isbetw((int)simage->idrc.idnbits,1,7))
    {
        bitcntr = 0;                /* Init bit counter */
        bitbuffer = 0;              /* Init buffer */
        destcntr = 4;               /* Count numbers converted */

        memcpy(destptr,Inbuf,VAXOFFSET); /* Copy 4 bytes(VAX rec header)*/

        for(i=VAXOFFSET; i<simage->idrc.idnppl;i++)
        {
            for(j=0; j<8; j++)
            {
                if(Inbuf[i] & bit[j])
                    bitbuffer |= bit[bitcntr]; /* Set bit */
            }
        }
    }
}

```

```

else
    bitbuffer &= ~bit[bitcntr];    /* Clear bit */

    if(++bitcntr == simage->idrc.idnbits)
    {
        /* Have a complete number, so lets store it
           reset counter and store buffer in memory */

        destptr[destcntr] = scale( bitbuffer,
                                   simage->idrc.idnbits,
                                   8 );

        bitcntr = 0;  bitbuffer = 0;  destcntr++;

        /* if we have amount of pixels for one row, turn the
           updated pointer to the ISIFS data storage.  */

        if(destcntr == simage->idrc.idnppl + 4)
            return( (void *) (destptr + newrecbsize));
    }
}

/* OOPs, this kinda file is not in my repetoire, Return OOPS */
    NotifyUser(-1,-1,"DEBUG","ERROR - Cannot do this kindof file\n");
}

```

```

void *short_convert(lnbuf,destptr,recl_b,newrecbsize)
unsigned char *lnbuf;
signed short *destptr;
int recl_b, newrecbsize;
{
    signed short bit[16];          /* Used to set bits */
    signed short bitbuffer;        /* A buffer to build 1 byte of data */
    int bitcntr;                  /* A bit counter */
    int i,j, k, destcntr;          /* Incrementers */
    unsigned int powit;            /* use to duplicate pow(x,y) */

    /* Load bit - note, wanted to pow() but double to uns char is nasty */

    powit = 1;
    for (i=0; i<16; i++)
    {
        bit[i] = (signed short) powit;
        powit *= 2;
    }

    /* ***** */
    /* Super Fast image loading * /
    /* signed 16-bit, copy memory byte for byte * /
    /* ***** */

    if ( (simage->idrc.idmode == 1 && simage->idrc.idnbits == 16) )
    {
        memcpy(destptr,lnbuf, recl_b);

        return( (void *) (destptr + (newrecbsize/2)) );
        /* Return updated pointer */
    }

    /* ***** */
    /* Slow image loader, 1-7 bits * /
    /* if (nbits >= 1-bit and <= 15) && mode is unsigned integer * /
    /* ***** */

    if( simage->idrc.idmode==1 && isbetw((int)simage->idrc.idnbits,1,15))
    {
        bitcntr = 0;                /* Init bit counter */
        bitbuffer = 0;              /* Init buffer */
        destcntr = 2;               /* Init destination cnt to*/
        /* to byte 4 */

        /* Determine the length of data/row in bytes by multiplying the
        number of pixels per row by 2 (i.e.,the sizeof a short */

```

```

memcpy(destptr,lnbuf,VAXOFFSET);      /* Copy 4 bytes (VAX record header
(  */

for(i=VAXOFFSET; i<recl_b;i++)
{
    for(j=0; j<8; j++)
    {
        if(lnbuf[i] & bit[j])
            bitbuffer |= bit[bitctr];      /* Set bit */
        else
            bitbuffer &= ~bit[bitctr];     /* Clear bit */

        if(++bitctr == simage->idrc.idnbits)
        {
            /* check to see if sign bit is flipped, if so, flip the
               rest of the bits in the 16-bit bitbuffer */

            if(bitbuffer & bit[bitctr-1])
                for (k = bitctr; k <16; k++) bitbuffer |= bit[k];

            /* reset counters an store buffer in memory */

            destptr[destctr] = scale( bitbuffer,
                                      simage->idrc.idnbits,
                                      16 );
            bitctr = 0; bitbuffer = 0; destctr++;

            /* if we have amount of pixels for one row, return the
               updated pointer to the ISIFS data storage.  */

            if(destctr == simage->idrc.idnppl + 2)
                return( (void *) (destptr + newrecbsize/2));
        }
    }
}

/* OOPS, this kinda file is not in my repetoire, Return OOPS */
NotifyUser(-1,-1,"DEBUG","ERROR - Cannot do this kindof file\n");
}

```

```

void *float_convert(Inbuf,destptr,recl_b,newrecbsize)
unsigned char *Inbuf;
float *destptr;
int recl_b;
int newrecbsize;
{
    unsigned long bit[32];          /* Used to set bits */
    signed long bitbuffer;          /* A buffer to build 1 byte of data */
    unsigned long powit;
    int bitcntr;                    /* A bit counter */
    int i,j,k,destcntr;            /* Incrementers */

    /* Load bit - note, wanted to pow() but double to uns char is nasty */

    powit = 1;
    for (i=0; i<32; i++) {
        bit[i] = powit;
        powit *= 2;
    }

    /******
    /*      Super Fast image loading                                */
    /*      signed 32-bit float, copy memory byte for byte          */
    /* NOTE: DO not read such a file produced by the VAX, the      */
    /*      float on VAX is not equivalent to the ASCII float.      */
    /******

    if ( simage->idrc.idmode == 2 && simage->idrc.idnbits == 32 )
    {
        memcpy(destptr,Inbuf, recl_b);
        return( (void *) (destptr + (recl_b/sizeof(signed long))) );
    }

    /******
    /*      Medium speed image loading                                */
    /*      unsigned or signed 32-bit, copy memory byte for byte    */
    /******

    if ( simage->idrc.idmode == 0 && simage->idrc.idnbits == 32 )
    {
        /** Insert Code Here **/

        NotifyUser(-1,-1,"DEBUG","not implemented\n");
        /* Return updated pointer */
    }

    /******
    /*      Slow image loader                                        */
    /* if (nbits >= 9-bit and <= 15) && mode is unsigned integer, or */

```

```

/* if (nbits >= 17-bit and <=31-bit && mode is signed integer.          */
/*******/

if((simage->idrc.idmode==0 && isbetw((int)simage->idrc.idnbits,9,16)) ||
   (simage->idrc.idmode==1 && isbetw((int)simage->idrc.idnbits,17,31)) )
{
    bitcntr = 0;                      /* Init bit counter */
    bitbuffer = 0;                   /* Init buffer */
    destcntr = 1;

    memcpy(destptr,lnbuf,VAXOFFSET); /* Copy 4 bytes (VAX record header
( */

    for(i=VAXOFFSET; i<recl_b;i++)
    {
        for(j=0; j<8; j++)
        {
            if(lnbuf[i] & bit[j])
                bitbuffer |= bit[bitcntr]; /* Set bit */
            else
                bitbuffer &= ~bit[bitcntr]; /* Clear bit */

            if(++bitcntr == simage->idrc.idnbits)
            {
                /* check to see if sign bit is flipped, if so set the
                   the bits in the rest of the 32-bit bitbuffer. */

                if ( (bitbuffer & bit[bitcntr-1]) && simage->idrc.idmode == 1)
                    for (k = bitcntr; k <32; k++) bitbuffer |= bit[k];

                /* reset counter an store buffer in memory */

                destptr[destcntr] = (float) bitbuffer;
                bitcntr = 0; bitbuffer = 0; destcntr++;

                /* if we have amount of pixels for one row, turn the
                   updated pointer to the ISIFS data storage. */

                if(destcntr == simage->idrc.idnppl + 1)
                    return( (void *) (destptr + newrecbsize/4));
            }
        }
    }
}

/* OOPS, this kinda file is not in my repetiore, Return OOPS */

NotifyUser(-1,-1,"DEBUG","ERROR - Cannot do this kindof file\n");
}

```

```

int querySIF( fn,nbits,nppl,nlins,ncols,nrows,nbnds,nsbnds,mode,isif_type )
char  *fn;           /* pointer to the file name */
int    *nbits,        *nppl,        *nlins,
        *ncols,        *nrows,        *nbnds,
        *nsbnds,        *mode,        *isif_type;
{
    ISIFS asimage;      /* structure of internal SIF */
    int  recl_b;        /* record length in bytes */

    /* Open the file and assign the return to a global pointer */

    simage = &asimage;

    fptr = fopen(fn,"rb");
    if( fptr == NULL )
        NotifyUser(-1,-1,"DEBUG","ERROR, cannot open file");

    /* Load the initial record, calculate the size of the records,
       and load the identification record. */

    load_initial_record();
    recl_b = ( (int)simage->idrc.idnbtsrc/8) + VAXOFFSET;
    load_ident_record( recl_b );

    /* Lets now assign the variables passed in as parameters */

    *nbits  = (int)simage->idrc.idnbits;
    *nppl   = (int)simage->idrc.idnppl;
    *nlins  = (int)simage->idrc.idnlins;
    *ncols  = (int)simage->idrc.idncols;
    *nrows  = (int)simage->idrc.idnrows;
    *nbnds  = (int)simage->idrc.idnbnds;
    *nsbnds = (int)simage->idrc.idnsbnds;
    *mode   = (int)simage->idrc.idmode;

    /* Now determine the data type the ISIF will use to represent the
       data. */

    *isif_type = decide_type( (int)simage->idrc.idmode,
                              (int)simage->idrc.idnbits );
    if( isif_type == 0 )
        NotifyUser(-1,-1,"DEBUG","Cannot assign a type. \n");
    return(0);
}

```

```

/*-----*/
/*

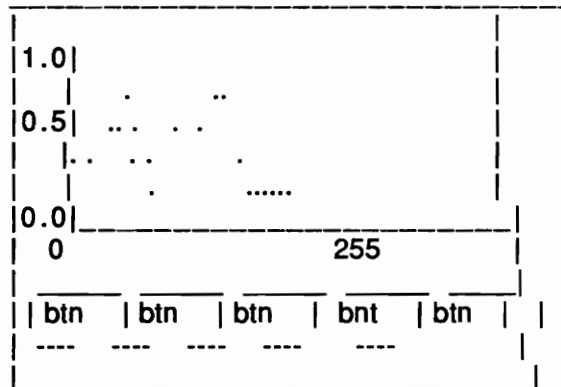
```

Module Name: HISTFORM.C

Purpose: To present the user a histogram of color values from a selected region of a bitmap.

Procedures: void histogram()
static void InitHorz()
static void drawit()

Description: First create a form that has an area to draw a histogram. Place a row of buttons on the bottom of the form. These buttons allow the user to display various types of histograms.



Hardware: Compaq Despro 386/20

Compiler: Microsoft C version 5.1

Linker: cl /c /AL %1.c
cl main binform /link met_bd1d mw_d1dmc
menuetml mnutlml /ST:4000

Requires MetaWindows 3.6 and
Menuet 1.7E

Author: Karl Melder - Fall 1990

Future Updates:

(1) Pass in array of histogram values into histogram(). The values in this array must be scaled to be properly displayed on the screen.

(2) Attach functionality to row of buttons
on the bottom of the form

```

/* ----- */

#include "stdlib.h"
#include "stdio.h"
#include "string.h"

#include "GRconst.h"
#include "GRports.h"
#include "GReXtrn.h"

#include "menuet.h"

#define nBUTT 5

ob_desc buttons[nBUTT] =
{
    { 0, 0, 0, 0, 0, 0, 0, 0, 'B', 0, "Point", NULL, 0, 0, 0, 0, NULL },
    { 0, 0, 0, 0, 0, 0, 0, 0, 'C', 0, "Bar", NULL, 0, 0, 0, 0, NULL },
    { 0, 0, 0, 0, 0, 0, 0, 0, 'E', 0, "Smooth", NULL, 0, 0, 0, 0, NULL },
    { 0, 0, 0, 0, 0, 0, 0, 0, 'M', 0, "Rescale", NULL, 0, 0, 0, 0, NULL },
    { 0, 0, 0, 0, 0, 0, 0, 0, 'Q', 0, "Quit", NULL, 0, 0, 0, 0, NULL }
};

fm_desc butFm =
{
    0, 0, 0, attrDgBox, IS_INIT, -1,
    NULL, 0, 0, 0, 0, nBUTT, buttons,
    0, NULL, 0, NULL
};

/* ----- */

#define WWIDTH 40          /* window width in characters, choose
                           only that can be evenly divided by
                           the number of buttons */

#define HIST_WIDTH 32      /* Width of histogram in characters */
#define HIST_HEIGHT 10    /* Must be divisible by 5 */

char *vert_grads[5] = { "0.00", "0.25", "0.50", "0.75", "1.00" };
char *horiz_grads[5] = { "0", "63", "127", "191", "255" };

void histogram ( );        /* main entry - creates the form */
static void InitHorz ( );  /* create a row of buttons */
static void drawit ( );    /* draw the histogram on the form */

```

```
extern char *fnt_buf;          /* The additional font for drawing  
the histogram */
```

```

/*-----*/
/* Lets make a histogram */
/*-----*/

void histogram( void )
{
    int          i,          /* loop counter */
              done;          /* done switch */
    attr_blk     *ab;        /* pointer to attribute block */

    /* Hide the cursor before doing anything */

        HideCursor();

    /* Lets design the form here */

        ab = GetAttrBlk(attrDgBox);
        ab->bg_clr = ColorPalette[7];
        ab->fg_clr = ColorPalette[0];
        ab->thk = 1;
        ab->pat = 0;

    /* Set up form size, attributes, and add buttons */

        InitHorz();

    /* Open the form and draw the histogram in it */

        OpenForm( &butFm, -1, -1 );
        drawit( fnt_bufr );
        /* scale_it(hist[]);
           draw_it(scaled_hist[]);
        */

        ShowCursor();          /* Now allow user to select something */
        done = 0;
        while( ! done )
        {
            SelectOb( buttons, nBUTT, 1 );
            break;
        }

    /* Lets clean up */

        CloseForm( &butFm );
}

```

```

/*-----*/
/*      Set up the form dimension and the buttons      */
/*-----*/

static void InitHorz( void )
{
    int          i,                /* counter */
              dx,                /* change in x */
              dy;                /* change in y */
    attr_blk     *ab;             /* pointer to attribute block */
    point        topleft;         /* Upper left corner of form */

    /* Compute first button's rect, adjust to include border, then
       offset its position to (0,0). */

    dy = charH + 2;
    dx = ((WWIDTH/nBUTT) + 2) * charW;
    SetRect( &buttons[0].area, 0, 0, dx-1, dy-1 );
    BxWork2Area( &buttons[0].area, &buttons[0].area, attrBut );
    OffsetRect( &buttons[0].area, -buttons[0].area.Xmin,
               -buttons[0].area.Ymin );

    /* Repeat for remaining buttons, but offset each button to about
       the previous button to the left. */

    for( i=1; i<nBUTT; i++ )
    {
        dx = (strlen(buttons[i].label) + 2) * charW;
        SetRect( &buttons[i].area, 0, 0, dx-1, dy-1 );
        BxWork2Area( &buttons[i].area, &buttons[i].area, attrBut );
        OffsetRect( &buttons[i].area,
                   buttons[i-1].area.Xmax-buttons[i].area.Xmin,
                   -buttons[i].area.Ymin );
    }

    /* Run through the button list and set remaining
       fields, and inset each rect by border thickness */

    ab = GetAttrBlk(attrBut);
    ab->thk = 1;

    for( i=0; i<nBUTT; i++ )
    {
        InitObFields( buttons+i, 1, 0, obButton, attrBut,
                     IS_EXITABLE|IS_SELECTABLE|IS_INVERTIBLE, LINK_HZ );
        BxArea2Work( &buttons[i].area, &buttons[i].area, attrBut );
    }
}

```

```
/* Set the form's area rect before proceeding */  
  
topleft.X = buttons[0].area.Xmin;  
topleft.Y = buttons[0].area.Ymin -  
            ( dy + (HIST_HEIGHT * charH) + (2 * charH) + 2 );  
  
SetRect( &butFm.area, topleft.X, topleft.Y ,  
         buttons[nBUTT-1].area.Xmax, buttons[nBUTT-1].area.Ymax );  
  
BxWork2Area( &butFm.area, &butFm.area, attrDgBox );  
}
```

```

/*-----*/
/* Draw the histogram lines and ruler marks */
/*-----*/

static void drawit( char *fnt_bufr )
{
    int          temp_pos,          /* temporary position */
               cnt_to_8,           /* count to 8 */
               i,                  /* counter */
               fnt_hgt,            /* font height in pixels */
               fnt_wdth,           /* font width in pixels */
               intervals,          /* spacing in pixels */
               offset;             /* indentation of x,y origin
                                   to 0 values on histogram are
                                   not on the lines */
    point        hst_origin;        /* origin point for histogram */

    /* Set of origin for the histogram, and set up pen color and size */

    hst_origin.X = buttons[0].area.Xmin + ((WWIDTH-
HIST_WIDTH)/2+1)*charW;
    hst_origin.Y = buttons[0].area.Ymin - (2 * charH + 1);

    PenNormal();
    PenColor(0);
    PenSize(2,2);

    /* Draw the horizontal axis, then the vertical axis */

    offset = charW;                  /* add a little so any data on
                                   w/value 0 will be seen */
    MoveTo( hst_origin.X, hst_origin.Y );
    LineRel( HIST_WIDTH * charW + offset, 0 );

    MoveTo( hst_origin.X, hst_origin.Y );
    LineRel( 0, -(HIST_HEIGHT * charH + offset) );

    /* Let draw the marks on the Horizontal axis at 1 charW intervals
    such that we have the following gradiations: 0, 63, 127, 191, 255 */

    cnt_to_8 = 8;
    temp_pos = hst_origin.X;

    for( i = 0; i <= HIST_WIDTH; i++)
    {
        /* Move pen to correct position to start writing ruler marks */
        temp_pos += offset;
        MoveTo(temp_pos, hst_origin.Y);
    }
}

```

```

/* Draw a little mark at each charW. Draw larger ones
   at every 8th position */

    if(cnt_to_8 == 8) {                /* if 8th position */
        LineRel(0, charH/4);          /* draw a large ruler mark */
        cnt_to_8 = 0;
    }
    else LineRel(0, charH/8);          /* else draw a small ruler mark */

    cnt_to_8++;
}

/* Let draw the ruler marks on the vertical axis at interval, such that
   we have the following gradations: 0.0, 0.25, 0.5, 0.75, 1.0 */

temp_pos = hst_origin.Y - offset;
intervals = (HIST_HEIGHT * charH)/ 4;

for( i = 0; i < 5; i++)
{
    MoveTo(hst_origin.X, temp_pos );    /* position the pen */
    temp_pos -= intervals;              /* set next position */
    LineRel(-charH/4, 0);               /* draw a ruler mark */
}

/* Set to a small font to be used with the histogram, but save the
   original fonts width and height. */

fnt_hgt = charH;
fnt_wdth = charW;
SetCurFont( (fontRec *) fnt_buf );

/* Now lets draw the ruler numbers on horizontal axis */

/* Align text to be position below and centered the ruler mark */
TextAlign( alignCenter, alignTop);
TextMode(0);

temp_pos = hst_origin.X + offset;
MoveTo(temp_pos, hst_origin.Y + fnt_hgt/2);

for( i = 0; i < 5; i++)                /* Draw 5 numbers */
{
    DrawString( horiz_grads[i] );
    temp_pos += fnt_wdth*8;
}

```

```

        MoveTo(temp_pos, hst_origin.Y + fnt_hgt/2);
    }

/* Now lets draw the ruler numbers on vertical axis */

/* Align text to be positioned to left and middle of ruler mark */
TextAlign( alignRight, alignMiddle);
TextMode(0);

temp_pos = hst_origin.Y - offset;
intervals = (HIST_HEIGHT * fnt_hgt)/ 4;

for( i = 0; i < 5; i++)          /* Draw 5 numbers */
{
    MoveTo(hst_origin.X - fnt_hgt/2, temp_pos );

    DrawString( vert_grads[i] );
    temp_pos -= intervals;
}

/* Clean up before leaving */

SetMnFont();          /* Set back to main font */
}

/* SIF structures */
/* The following structures are based on the documentation of
SIF files as described in the Gipsy User's Guide, pp. 101 - 104.
This may have to be updated as the understanding of the VAX
record structure increases, and the need to port SIF files
over to the VAX. Will SIF structure between PC gipsy and
gipsy be the same? */

typedef struct {
    long id0x01,          /* The 1st 4 bytes of VAX record */
        idnrclsfl,        /* Number of Records in the SIF file */
        idnbtsrc, /* Number of Bits in a record */
        id0x0C,
        id0x00,
        idb_nrclsfl,      /* Duplicate of idnrclsfl */
        idb_0x00;         /* Pad w/0's until EORecord */
} InitialSifRec;

typedef struct {
    long idVAX,           /* The 1st 4 bytes in VAX record */
        idusr1,           /* Unused */

```

```

    idusr2,          /* Unused */
    idsys1,          /* Unused and reserved */
    idsys2,          /* Unused and reserved */
    idnbits,         /* Number of bits/pixel */
    idnppl,          /* Number of pixels/line */
    idnlins,         /* Number of lines in an image */
    idhrcls,         /* Relative Horiz pixel dimension */
    idvrcls,         /* Relative Vert pixel dimension */
    idnscrs,         /* Number of descriptor records in file */
    idnql,           /* Number of quantized grey levels */
    idnwds,          /* Number of words per record */
    idncols,         /* Number of columns in each subimage */
    idnrows,         /* Number of rows in each subimage */
    idmin,           /* Minimum grey tone over all bands */
    idmax,           /* Maximum grey tone over all bands */
    idnbnds,         /* Total number of bands in an image */
    idnsbnds,        /* Number of symbolic (map) bands */
    idmode,          /* Data mode indicator */
    idver;           /* Version number of sif format */
} IdentSifRec;

```

```

typedef struct {
    InitialSifRec inrc;
    IdentSifRec idrc;
    char *history_ptr;
    char *data_ptr;
} ISIFS;

```

```

/*-----*/
/*  include "PROJECT.H"                                */
/*  definitions and structures for windowing environment */
/*  for PC-Gipsy                                        */
/*-----*/

```

```

/* define the type of files we can load */

```

```

#define PCX  1
#define PCC  1
#define SIF  2
#define BIN  3
#define RLE  4
#define IMG  5
#define MSP  6

```

```

/* define colors for EGA, VGA, and SVGA compatible (default setting).
   If it is necessary to go monochrome, must change to different scale */

```

```

#define BLACK      0
#define BLUE       1
#define GREEN      2
#define CYAN       3
#define RED        4
#define MAGENTA    5
#define BROWN     6
#define LIGHTGREY  7
#define DARKGREY   8
#define LIGHTBLUE  9
#define LIGHTGREEN 10
#define LIGHTCYAN  11
#define LIGHTRED   12
#define LIGHTMAGENTA 13
#define LIGHTYELLOW 14
#define LIGHTWHITE 15

```

```

/* The following are the Monochrome colors

```

```

/*-----*/
/*  Structure associated with each window.                */
/*  Used for maintaining multiple image windows           */
/*-----*/

```

```

typedef struct
  image_window
  {

```

```

rect  viewA,          /* Global coord. of viewing window */
      imagSrc,        /* Global coord. of source rect. */
      imagDst;        /* Global coord. of destination rect. */

rect  lcl_viewA;      /* Local coord. of viewing window */

rect  slctR;          /* select rectangle */
int    selected;      /* YES/NO if user has selected an area */

bitmap      imagMap;  /* Where the whole image is stored */

int    gbl_dx,
      gbl_dy,
      gbl_np,
      gbl_nb;
}
IMAGEWINDOW;

```

Source Code For Spawning Tasks

/* **** */

Programs: expl_spw.c and testb.c

Purpose: To have one task execute another in extended DOS mode.
Each task is compiled and linked in protected mode.

Functions: SYSTEM()

How to compile and link:

The "SPAWN" command does not work in extended DOS mode. One must choose the "SYSTEM" command, which has the added benefit of being ANSI standard.

Process:

- 1) compile expl_spw.c and testb.c using HC386 compiler
e.q., HC386p expl_spw.c
- 2) link using libraries and setting the memory location of the parent task. The value for MINREAL or MAXREAL is in 16 byte paragraphs. For RUN386 to be loaded requires 100K bytes, translating to a minimum n value of 6250.
e.q., 386link expl_spw -lib hce na -minreal 7000 -maxreal 7000
- 3) run the program using extended DOS environment
e.q., run386 expl_spw

/* **** */

```
#include "inc/stdio.h"
#include "inc/process.h"
#include "inc/stdlib.h"
```

```
void main()
{
    int retcode;

    printf("Attempt to spawn a task \n\n");
    if ((retcode = system("run386 testb.exp")) == 0)
        puts("The child process terminated normally.");
    else if (retcode == -1) {
        puts ("The child process was not started.");
    }
    else
    {
        puts ("The child process had an abnormal exit.");
        printf("retcode = %d\n",retcode);
    }
}
```

/*****

Programs: expl_spw.c and testb.c

Purpose: To have one task execute another in extended DOS mode.
Each task is compiled and linked in protected mode.

Functions: SYSTEM()

How to compile and link:

The "SPAWN" command does not work in extended DOS mode. One must choose the "SYSTEM" command, which has the added benefit of being ANSI standard.

Process:

- 1) compile expl_spw.c and testb.c using HC386 compiler
e.g., HC386p expl_spw.c
- 2) link using libraries and setting the memory location of the parent task. The value for MINREAL or MAXREAL is in 16 byte paragraphs. For RUN386 to be loaded requires 100K bytes, translating to a minimum n value of 6250.
e.g., 386link expl_spw -lib hce na -minreal 7000 -maxreal 7000

- 3) run the program using extended DOS environment
e.g., run386 expl_spw

*****/

```
#include "inc/stdio.h"
#include "inc/process.h"
#include "inc/stdlib.h"
```

```
void main()
{
    int retcode;

    printf("Attempt to spawn a task \n\n");
    if ((retcode = system("testb.exe")) == 0)
        puts("The child process terminated normally.");
    else if (retcode == -1) {
        puts ("The child process was not started.");
    }
    else
    {
        puts ("The child process had an abnormal exit.");
        printf("retcode = %d\n",retcode);
    }
}
```