Old Dominion University
ODU Digital Commons

Electrical & Computer Engineering Theses & Dissertations

Electrical & Computer Engineering

Spring 2004

Generic Performance Evaluation Tool Implementation for an Imaging Fourier Transform Spectrometer

James M. Mengert
Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/ece_etds

Part of the Computer Engineering Commons, Programming Languages and Compilers Commons, and the Software Engineering Commons

Recommended Citation
Mengert, James M.. "Generic Performance Evaluation Tool Implementation for an Imaging Fourier Transform Spectrometer" (2004). Master of Science (MS), Thesis, Electrical & Computer Engineering, Old Dominion University, DOI: 10.25777/7jm8-6b58
https://digitalcommons.odu.edu/ece_etds/431

This Thesis is brought to you for free and open access by the Electrical & Computer Engineering at ODU Digital Commons. It has been accepted for inclusion in Electrical & Computer Engineering Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.

GENERIC PERFORMANCE EVALUATION TOOL IMPLEMENTED FOR AN IMAGING FOURIER TRANSFORM SPECTROMETER

by

James M. Mengert
B.S. Computer Engineering, May 2001, Old Dominion University

A Thesis submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

COMPUTER ENGINEERING

OLD DOMINION UNIVERSITY
MAY 2004

Approved by:

[illegible]
Lee A. Belfore II (Director)

[illegible]
John W. Stoughton (Member)

[illegible]
James F. Leathrum, Jr. (Member)

ABSTRACT

GENERIC PERFORMANCE EVALUATION TOOL IMPLEMENTED FOR AN IMAGING FOURIER TRANSFORM SPECTROMETER

James M. Mengert
Old Dominion University, 2002
Director: Dr. Lee A. Belfore II

Evaluation software for analyzing general analytical model applications utilizing MATLAB is presented in this thesis. The focus is to provide an interactive environment for the evaluation of model applications relating to an Imaging Fourier Transform Spectrometer. The Generic Performance Evaluation Tool (GPET) was developed by researchers at Old Dominion University and NASA-Langley Research Center and was a NASA-Langley Research Center sponsored project. GPET is capable of producing a multitude of object-oriented input data for any number of individual model applications. The tool is also capable of creating user-defined input interfaces for the purpose of viewing and/or editing input object data, along with creating user-defined result interfaces for viewing evaluated object data in a textual format. Additionally, the tool can create various user-defined graphs to allow object data to be evaluated in a graphical format. With the model applications being designed within the context of the software, all of these capabilities provide the user with a very flexible tool for evaluating any analytical model application.

This thesis is dedicated to my wife, Paula, and our son Christopher

ACKNOWLEDGEMENTS

I would like to acknowledge the people who have helped me in this work. First, I would like to thank Dr. Lee A. Belfore II, for his support and direction in the development of this thesis. Second, I would like to thank Dr. John W. Stoughton, for his advice and guidance in the development of the software.

I would like to thank the most important people in my life. My wife, Paula, has provided an enormous amount of support throughout my graduate studies. My parents, Helen and Elroy, have always provided me with encouragement in all my endeavors. My sister, Margie, has constantly imparted a positive outlook on life. And finally, to our son Christopher, who, in his own way, has reminded me that there must always be time to play.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

SYMBOL	DESCRIPTION
API	Application Program Interface
CTRL	Control
DOS	Disk Operating System
FORTRAN	Formula Translation
GIFTS	Geostationary Imaging Fourier Transform Spectrometer
GPET	General Purpose Evaluation Tool
GUI	Graphical User Interface
MATLAB	Matrix Laboratory
NASA	National Aeronautics and Space Administration
SNR	Signal-to-Noise Ratio
V&V	Verification and Validation

CHAPTER I

INTRODUCTION

1.1. Overview

The term "model" can be used to represent just about any entity. The automotive industry uses computer visualization to help aid in bringing a concept into reality; architects build models of buildings to aid in their visualization of what the real building would look like at a scaled down level. In this thesis, analytical models are used to describe physical entities. Analytical models will be used within a contained environment referred to as the General Performance Evaluation Tool (GPET). The tool has the ability to evaluate any number of analytical models; however, in this thesis a specific analytical model will be evaluated to demonstrate the capabilities of GPET.

The analytical model used in this thesis is an instrument called GIFTS. Geostationary Imaging Fourier Transform Spectrometer (GIFTS) is a next generation satellite that will significantly improve 3- to 5-day weather forecasting. The goal of the GIFTS instrument is to retrieve temperature, water vapor, and wind sounding information, along with chemical composition information of the atmosphere from space [7]. Modeling is also used in this thesis to help describe the software tool that will be used to evaluate the end user model applications.

1.2. Motivation

This project began in the fall of 2000 as an undergraduate senior design project[1]. Part of the goal of the project was to develop a user-friendly Graphical User Interface (GUI) to support the development of an end-to-end model of an imaging infrared Fourier

[1] The journal model followed in the thesis is IEEE Transactions on Computer.

Transform Spectrometer [4]. The GUI shell was to have the ability to manage a number of model applications, be able to edit input values for each application, and save input and result data to a global database, where the user can evaluate the data through such means as spreadsheets. Even though the project was successful, many improvements were identified that could be added in future versions. These improvements include a more efficient manner to install model applications, the use of data objects for use within the model applications, the creation of interfaces for editing input data as well as viewing result data, and providing the ability to create various types of graphs as an alternative for evaluating application-specific data.

The research for this thesis evolved from a need to have a flexible system for creating/managing analytical models for scientific instrumentation. The project was funded by NASA-Langley Research Center to support instrument modeling in the GIFTS project.

1.3. Modeling

Models are used in many industries and most engineering disciplines. Models can take on many forms that most may not even realize. These forms include mathematical equations, a physical entity, or even a mere guiding mental image [8]. The four reasons for model development are that it is easier, less expensive, faster to analyze a model than the entity being modeled, and the entity may not exist [8].

In this thesis, conceptual models are presented to give an abstract understanding of the functionality behind the real software [9], but do not represent the software itself. A conceptual model is defined as having components that have not been clearly identified in terms of system-theoretic categories such as state, event, and function [9].

1.4. Graphical User Interface Design

The main reason the Graphical User Interface (GUI) exists is to give an efficient portal that employs visual cues for effectively using the software application. Before the GUI, users had to use keyboard entry to enter specific commands at a shell prompt[2] to accomplish such tasks as editing text files, formatting hard drives, or to launch applications. With the GUI, the user is presented with pushbuttons, radio buttons, and list boxes with appropriate actions to allow the user to make decisions based on the type of application is running.

On the surface, it may seem that a GUI is easy to design, develop, and implement. However, there are a number of design concepts that the GUI programmer needs to be aware of, and these concepts are discussed in this thesis.

1.5. Verification and Validation

Verification and Validation (V&V) is probably the most important process used in testing. V&V is used in software to help build quality in the end product by comprehensively analyzing software during each stage of software development (Verification), along with analyzing the completed product (Validation) [3].

The concept of V&V evolved during the late 1960s and into the 1970s as software use in military and nuclear power systems increased [3], where not having such a process could lead to disastrous consequences. Since then, V&V has found its way into such industries as automotive, airlines, medical, home electronics, military, and space.

[2] Command shell, application shell: A place to enter commands to access computer resources or respectively run a simulation.

1.6. MATLAB Programming Language

MATLAB is a high performance language for technical computing [5]. It is an interpreted language that uses either script or function files. MATLAB can be used in many contexts including math and computation, algorithm development, modeling, simulation, and prototyping, data analysis, scientific and engineering graphics, and application development (including GUI implementation) [5].

The foundation data element for computing in MATLAB is an array that is dimensioned implicitly, that often allows the user to solve many technical problems involving matrices and vector formulations in a fraction of the time then it would take to write a program in a compiled language solution such as C or FORTRAN [5]. The MATLAB system consists of five main sections. They are the MATLAB language, working environment, handle graphics, mathematical function library, and the Application Program Interface (API), that is used when translating MATLAB code to the C language.

1.7. Research Objective

The objective of this thesis research is to present a methodology for managing models, and to demonstrate the methodology using a specific model. The tool will provide the capability of examining model application behavior based on a given set of object-oriented input parameters.

Without a specific and controlled environment, the developer of model applications must add functionality to the model applications that may not necessarily be directly related to the focus of the application. Some of the problems to deal with when creating model applications include: no consistency in the manner of viewing/editing input data and reviewing result data, any data that needs to be collected must be performed within the application, the input and result data could be vulnerable to corruption, no central location for the collection of data among multiple applications, and the composition of multiple models. These are the problems that the mechanisms discussed are focused on solving. Therefore, to meet the research objective, the following goals need to be met:

1. Install and uninstall model applications within a shell environment.
2. Create unlimited input data objects for use in evaluating model applications.
3. Modify input data object values through the use of an interface.
4. Provide the viewing of result object data in either a graphing format or by the use of an interface.
5. Use result data objects from one application as an input data object into a different application.
6. Provide the user the ability to create user-defined interfaces for the viewing/editing of input data object values, and for the viewing of result data object values.
7. Provide the ability to store input data and result data values to an external database text file.

1.8. Thesis Organization

Chapter II presents background information on the mechanisms used to create the Generic Performance Evaluation Tool (GPET). Chapter III discusses software specifications of GPET. Chapter IV presents the detailed implementation of GPET. Chapter V describes the testing and validation of the software. Chapter VI discusses the summary and future works for GPET.

CHAPTER II

BACKGROUND

2.1 Introduction

In this chapter, foundational information is laid out to lead to the design, development and testing of the GPET software. The foundation for this research began as a senior design project as an undergraduate student. Part of the project involved establishing an environment in which all model applications related to a NASA-sponsored project named Geostationary Imaging Fourier Transform Spectrometer (GIFTS) can be tested. A brief background of GIFTS and the undergraduate project is presented in §2.2. The heart of the GIFTS instrument is the Michelson Interferometer. A brief description of the Michelson Interferometer along with its intended use within GIFTS is given in §2.3. The foundation for the design, development, and implementation for this research project is the modeling of the system. §2.4 discusses modeling in general, however, the specific models that were developed will be discussed in more detail in Chapter III. §2.5 provides an overview of the software development processes by discussing program analysis, design, and implementation that takes the conceptual model of the system and defines a process to implement the tool. In §2.6, some of the key concepts of Graphic User Interface Design will be discussed. In §2.7, aspects of the MATLAB programming language are presented, where comparisons are made between that language and C++. The chapter summary in §2.8 highlights general information within this chapter.

2.2 GIFTS

GIFTS is analogous to a three-dimensional video camera that records the time evolution of atmospheric structure by taking snapshots of the atmosphere in intervals of time [7]. The instrument will measure water vapor, wind sounding and the chemical composition of the atmosphere, and will be used to improve the accuracy of three- to five-day weather forecasting [7].

Part of the goal of the Fall 2000 senior design project included the development of an object-oriented GUI shell to manage the GIFTS-based model applications. The term 'model application' refers to the specific script files developed and run in MATLAB. Script files will be discussed in §2.5. Even though the undergraduate project was successful, other opportunities for improvement were apparent. One of the improvements is to develop and implement a way of installing the model applications into the shell, because the installation process using the original shell was quite cumbersome. Some other improvements to the original shell are the implementation of an interface creation tool, object-oriented data, and a plotting tool. The overall object of this project is to do research on ways to improve the software, then design, develop, and implement a new version of software.

2.3 The Michelson Interferometer

The heart of the GIFTS instrument is the Michelson Interferometer. Figure 2.1 illustrates the basic components of the Michelson Interferometer. The components include a beamsplitter, two mirrors, collimator, condenser, and a detector.

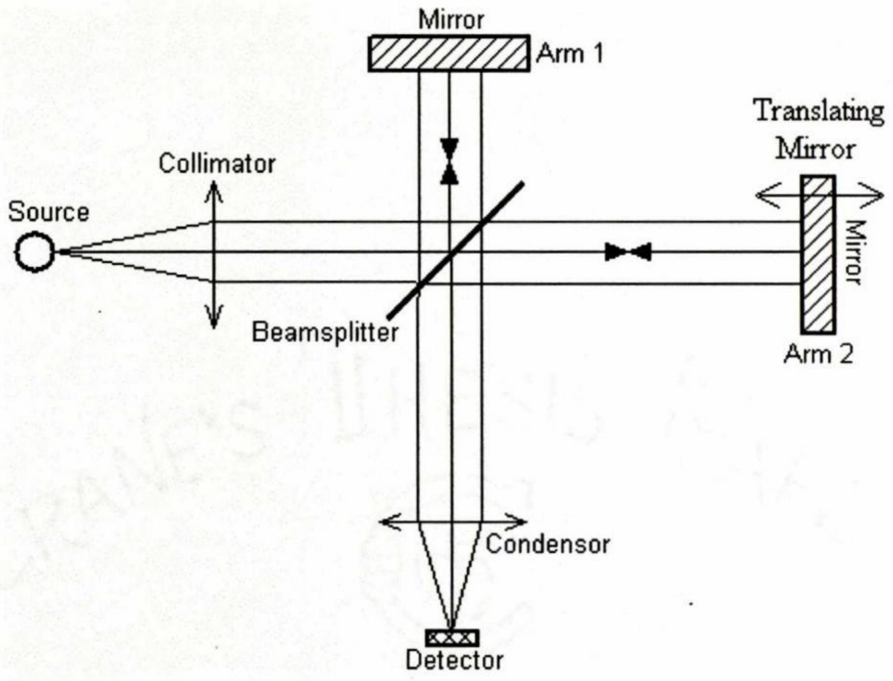


Figure 2.1 Michelson Interferometer

For the purposes of this discussion, the source is a monochromatic light. The source emits in all directions, so a collimator is used to render the source parallel. The source is then applied to a beamsplitter. Ideally, the beamsplitter allows half of the light to pass through, and reflect the other half. Fixing both mirrors, the two beams then travel to their respective mirrors on the arms of the interferometer. The returning beams are recombined at the beamsplitter, with the light waves interfering by the principle of superposition [10]. The interference of the waves will either be constructive or destructive. The recombined beam propagates to a condenser that focuses the beam on the detector. The detector measures the intensity at different mirror positions to form the interference pattern or interferogram [10].

The intended use of the Michelson interferometer is to measure broadband spectrums. The interferogram produced from the interferometer is the Fourier Transform of the spectrum. Interested readers can see [19] for the mathematical derivation.

2.4 Modeling

Models are used in most engineering disciplines because it is usually less expensive and requires less time to create models than the real entity. The purpose of any model is to facilitate analysis, either explicitly or implicitly [8]. A good model is one that is less complex than the entity it represents, but retains important salient characteristics.

Frequently, model development is a hierarchal process. The development begins with the conceptual model that is used when model components have not been clearly identified in terms of state, event, or function [9]. Once the conceptual model is completed, many paths can be taken to create a more system-theoretic model. These include the declarative, functional, constraint, and spatial models. Figure 2.2 illustrates the hierarchy of model design [9].

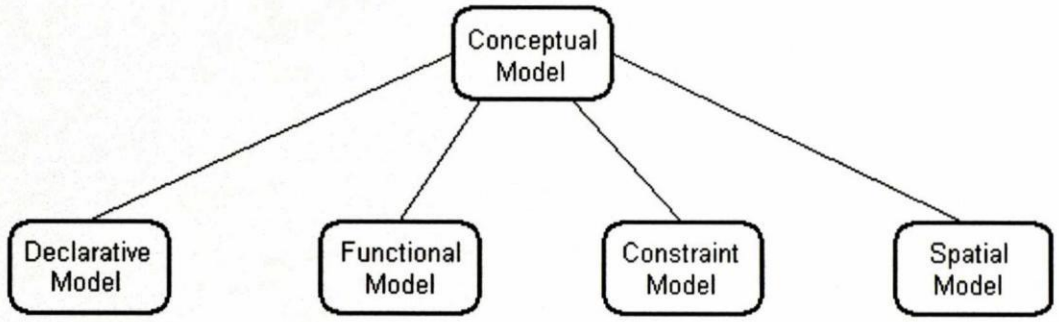


Figure 2.2 Model Development Progression

Further discussion of specific model development for this project will continue in Chapter III.

2.5 Software Development Stages

The GPET software tool is implemented using a three-step approach known as the Top-Down Design Process as follows: (1) Problem Analysis, (2) Problem Design, and (3) Program Implementation [1]. Seeing as it is practically impossible to solve any problem without knowing the nature of the problem, Problem Analysis defines what the software is suppose to do in terms of requirements. Problem analysis begins with a clear problem statement to guide the analysis process [1].

Stage two is the Program Design stage. In this stage, the original problem is broken down into smaller independent sub problems, where each of the sub problems is easier to solve than the original problem. More detail is added as the original problem is partitioned into smaller units. Once a sufficient amount of detail is added to the subproblem, an algorithm can be created to solve each of the subproblems [1].

The third stage is the Program Implementation stage. It is at this stage that a particular programming language is selected and used. The algorithms created in stage two are implemented using the MATLAB programming language.

The Top-Down Design Process is utilized throughout the software development cycle. At any point within the process problems may occur that may require reviewing the problem statement, and possibly making changes. Figure 2.3 illustrates the entire software development process, and also includes Verification and Validation.

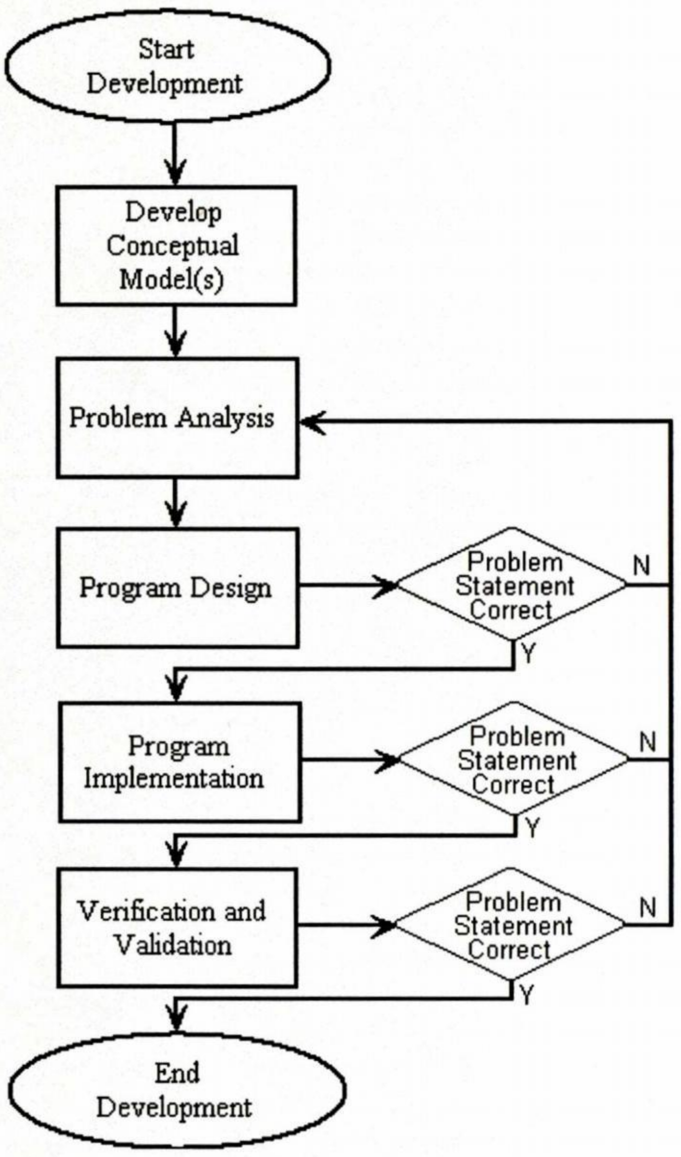

Start Development
Develop Conceptual Model(s)
Problem Analysis
Program Design
Problem Statement Correct
N
Y
Program Implementation
Problem Statement Correct
N
Y
Verification and Validation
Problem Statement Correct
N
Y
End Development

Figure 2.3 Software Development Process for GPET

2.6 GUI Design

During the creation of a GUI, it is easy to focus on what the developer wants. Therefore, it is essential that the developer remembers whom the interface is being created for, which is of course, the end user. GUI design should be thought of as an informal negotiation between the developer(s) and the end user(s). One of the most important aspects of efficient GUI Design is to involve the end user in the development process. The involvement should not apply specifically with the actual coding, but for such areas as layout of individual interface screens, high-level functions, and interface screen colors.

In addition to the aforementioned, do not constrain users to one type of choice on how to perform a particular task. For example, Microsoft Word® provides the user with various methods for saving files. The 'Save' operation could be performed from either a pull-down menu, or from an icon with an appropriate printer symbol. This allows different users to perform tasks in a manner in which they are accustomed, and be able to use that knowledge within another application.

The design should be kept simple while maintaining redundancy. Keep windows and dialogs clean and simple, otherwise the user could become confused [2]. Always make the user feel as though he/she knows what they are doing. Give the users simple choices when the application is first launched, and then make more functionality available as more windows come into focus.

In addition to the three concepts noted above, other GUI considerations are important. One is the ability to understand the behavior of people [2]. This became an issue for this project. At one point in the GPET development process, an interface was being created to allow the user to plot variables. Originally, the user had to recall specific variables from his model application before the variables could be plotted. The user recommended that instead, a list of all possible variables could be presented, allowing the user to recognize the specific variables to be plotted. The lesson learned in this example is that the user would rather recognize what is needed rather than trying to recall what is needed.

Another consideration is that the developers need to be aware of the context required at a particular time and also accepted practices for various icons [2]. While not an issue in this project, being aware of such things as placing a pushbutton object onto a window with a particular picture within the object could pose a problem. For example, placing a picture of an envelope on a pushbutton object may mean 'send this email now' to one user, while another user may understand it to mean 'save the email to a file'.

Clarity is defined as 'being free from obscurity and easy to understand'. When creating applications, the developer must be very clear when labeling pushbuttons, radio buttons, and the like. The developer needs to be careful of using words that have similar meanings. Using 'Product' on one pushbutton and 'Merchandise' on another pushbutton may confuse the user [2]. One possible remedy to avoid confusion when labeling objects is to create and use a table of reserved words [2]. The table could contain the reserved word, along with its meaning and behavior, and possibly whether it should appear on a pushbutton and/or pull-down menu, and/or a shortcut keystroke.

One last consideration here is to provide visual or audible feedback to the user. If selecting a particular item within an interface may delete a file, then it would be in the users' best interest to be warned. There are areas within the GPET system that provide such feedback.

2.7 The MATLAB Language

The MATLAB language is an interpretive language that shares features with other higher-level languages such as C++, but includes many differences as well. MATLAB uses many of the same reserved words such as 'if', 'for', and 'else', although the precise syntax differs. It also makes use of the class constructor to allow the encapsulation of attributes, and only allows methods of the class to act upon the attributes, keeping them contained. Furthermore, MATLAB has optional libraries that may need to be purchased that will compile the MATLAB code into object code and ultimately an executable.

The MATLAB language uses two forms of files: script and function files. Script files cannot accept input arguments or return output arguments, and only operate on variables within the user workspace. Also, variables from a script file persist within the workspace after running the file. In contrast, functions can accept input arguments, produce output arguments, and internal variables are local to the function. If a variable is shared among several functions, that particular variable must be declared GLOBAL within all functions that need to share the value within the variable.

Table 2.1 illustrates similarities and differences between the MATLAB and C++ programming languages. Some of the similarities include the use of class constructors, the 'class' reserved word, the use of inheritance, and the use of private methods and attributes. Differences include the use of a specific directory for the class, the directory name being the same as the class name, and the use of specific directories for subclasses, all of which are required in MATLAB. Also, MATLAB does not use the reserved KEYWORDS 'Private' and 'Public'. Instead, MATLAB reserves a private subdirectory for each class that will contain private methods.

Table 2.1 MATLAB/C++ Class Comparison

	MATLAB	C++
Specific directory for class	Yes	No
Class constructors	Yes	Yes
Class destructors	No	Yes
'Class' word reserved	Yes	Yes
Directory name same as classname	Yes	No
Executable created	No	Yes
Uses Inheritance	Yes	Yes
Specific directory for subclasses	Yes	No
Private methods and attributes	Yes	Yes
'Private' word reserved	No	Yes
'Public' word reserved	No	Yes

The 'varargin' is a variable used in MATLAB, and is a variable length input argument list (cell array). The cell array contains the optional arguments used by a function. When used, the 'varargin' variable must be the last input argument used by the function. For example, the function

function myplot (x, varargin)

shows 'x' as the first input argument, and 'varargin' as the last input argument. By using the 'varargin' argument within a class (or any function), the class will have a number of 'case' (or IF) statements to choose from, and the number of statements will depend on how many input arguments are within the 'varargin' variable, along with the number of arguments listed before the 'varargin' variable. Each statement will implement a particular constructor. The format is '@*classname/private/method_name*.m'. In this manner, the method has scope only within the '*classname*', which means that the method can be called by any method within the '*classname*' directory; however, it cannot be called from the MATLAB command line or by methods outside of the class directory [5]. In addition to the above, the file must be a function file (not script), and the function name must be *class_name* [5]. An example of the format in which a class in defined in MATLAB is shown in Figure 2.4

A particular constructor in C++ is called depending on the input argument(s), if there are any. In C++, all methods and attributes are public by default. If any method or attribute for a class must be private, then the reserved word 'private' is used within the class.

```
function object = classname(obj_name, varargin)
%
switch nargin
%
case 1
  Case 1 statements
  obj = class (obj_name, obj_type, obj)
case 2
  Case 2 statements
  obj = class (obj_name, obj_type, obj)
case n
  Case n statements
  obj = class (obj_name, obj_type, obj)
otherwise
  error ('Wrong number of input parameters')
end
```

Figure 2.4 Example MATLAB Class

Inheritance is also used in MATLAB, where child classes are allowed to inherit attributes and methods from its parent class. The constructor function for a class that inherits the behavior of another has two special characteristics. First, the constructor calls the constructor function for the parent class to create the inherited fields. Second, the calling syntax for the class function is slightly different, reflecting both the child and parent classes [5]. The use of inheritance implies that any object that belongs to a child class will have the same fields as the parent class plus any additional fields specifically for the child class. Also, any methods associated with objects of the parent class can operate on objects of the child class. However, methods that operate on objects of the child class cannot access fields within objects of the parent class. In Chapter IV we will look at inheritance within the GPET software. Figure 2.5 illustrates a partial Parent/Child class in MATLAB.

Inheritance is used in C++ much the same as it is used in MATLAB. In C++, child and parent classes are referred to as base (parent) and derived (child) classes [6]. Derived classes inherit all public methods and attributes from the base class. Any private methods and/or attributes in the base class cannot be inherited. In other words, inheritance does not imply access [6].

```
function object = inferiorclassname (obj_name, varargin)
%
inferiorto ('superiorclassname')
switch nargin
%
case 1
 Case 1 statements
 obj = class (obj_name, obj_type, obj)
case 2
 Case 2 statements
 obj = class (obj_name, obj_type, obj)
case n
 Case n statements
 obj = class (obj_name, obj_type, obj)
otherwise
 error ('Wrong number of input parameters')
end

function object = superiorclassname (input argument)
%
superiorto ('inferiorclassname')
%
superiorclassname.value01 = input argument value 01
                              .
                              .
                              .
 superiorclassname.value n = input argument value n
 object = class (obj, 'object_name')
 end
```

Figure 2.5 Parent/Child Class in MATLAB

2.8 Chapter Summary

This chapter began with a discussion of how this research came about. First, a brief introduction to the GIFTS related research was presented. Next, a discussion of general modeling was presented along with the hierarchal approach to modeling. The modeling discussion was followed by the software development stages that were involved in taking the tool from a problem statement to implementation. Next, background on the Graphic

User Interface (GUI) design was presented. Issues discussed included user perspective, clarity, and providing visual and audible feedback to users. MATLAB programming concepts were then discussed. Finally, a comparison of the development of classes and inheritance between MATLAB and C++ were briefly discussed along with an example of a class design in C++ as well as base and derived class examples in the same language.

CHAPTER III

SOFTWARE SPECIFICATIONS AND OVERVIEW

3.1 Introduction

GPET was developed for the purpose of evaluating the GIFTS instrument. The Top-Down Design technique was used in the overall development of GPET. The three phases of the Top-Down technique are (1) Problem Analysis, (2) Program Design, and (3) Program Implementation. In §3.2, these three phases will be discussed in more detail. The foundation to the design of this software is three types of conceptual models, that collectively, represent GPET, and §3.3 discusses these models. Following the Top-Down discussion, overall generic software architecture is developed and discussed. §3.4 illustrates the generic architecture that will be implemented in Chapter IV. §3.5 is the chapter conclusion.

3.2 Software Development Stages

There are three phases in the Top-Down Design process. In this section, the three phases will be discussed in detail. First, the Problem Analysis phase will be discussed, where a problem statement has been created, and used as the requirements in further development of GPET. Second, the Program Design phase will break down the problem statement into smaller subproblems. In breaking down a large problem into many smaller subproblems, then solving the smaller problems, in essence, the large problem is being solved. Once all large problems have been broken down into its smallest component, an algorithm can be developed to solve each subproblem. Third and finally, in the Program Implementation phase, the algorithms developed in phase two can be implemented in MATLAB.

3.2.1 Problem Analysis

The Problem Analysis phase begins by developing a problem statement. The problem statement is essentially a set of high-level requirements that state what the software is required to do. These requirements are the goals that were discussed in §1.7 (Research Objective).

There are many issues to deal with when looking at the problem statement in greater detail. Some of the more important issues include making the installation and un-installation process as user-friendly as possible, the creation of actual objects for both input and result data, and for providing an environment where users have the ability to view the results of data in either a textual interface format and/or graphical environment.

3.2.2 Program Design

The problem statement previously presented represents program functionality at a high level. The next phase is to take the problem statement and break each point down into smaller and smaller subproblems. As each original problem is sub-divided further smaller problems, each smaller problem will be much easier to solve. Once each of the smaller problems is sufficiently broken down, algorithms can be created to solve the smaller problems.

As an example of taking a large problem and dividing it up into subproblems, item one from the problem statement refers to the ability to install and uninstall model applications. When looking at the installation part of the problem, subproblems evolve. One of the subproblems would be to test the environment to see if an application with the same name exists. Another subproblem would be to ensure that each application has its own subdirectory to store not only the application files, but also all input and result interfaces that are created for the application, as well as any graphing plots for the application. There is also another subproblem of keeping track of experiment and result objects that are created, and then deleted, for a particular application, so that the environment can conserve memory and file space. Still another subproblem is the way of keeping track of the values of certain variables that are needed by all applications, but may have different values. These are examples of taking a larger problem of being able to install an application, and creating subproblems with more detail, to solve. In this situation, the program design is the process of taking a requirement of installing and uninstalling model applications from the problem statement, and developing smaller problems that are easier to solve than the original.

3.2.3 Program Implementation

The third software development phase takes the work that was accomplished in phase two (Program Design) and implements that work into a particular programming platform. As was stated before, the programming platform used to implement GPET is MATLAB. The reason for using MATLAB is two-fold. First, the end user developed the original GIFTS model application in MATLAB, and second, the predecessor to GPET was developed using the MATLAB platform.

3.3 Various Models

In Chapter II, modeling took a hierarchal approach that begins with the conceptual model. Fundamental to this thesis are three conceptual models presented in this subsection that become the foundation that represents the software system being created. The three conceptual models are known as the Compositional Model, the Operating Systems Model, and the Shell Model. These models were devised by the author to provide an understanding on how the tool should function. From the conceptual models, the GPET model (§3.3.4) is developed by using characteristics from the three conceptual models. These models are discussed in more detail in the following sections.

3.3.1 Compositional Model

The Systems Model is developed by first creating individual component models, then constructing the Systems Model by cascading individual component models. The Component Architecture that was developed and incorporated into the Systems Model is shown in Figure 3.1.

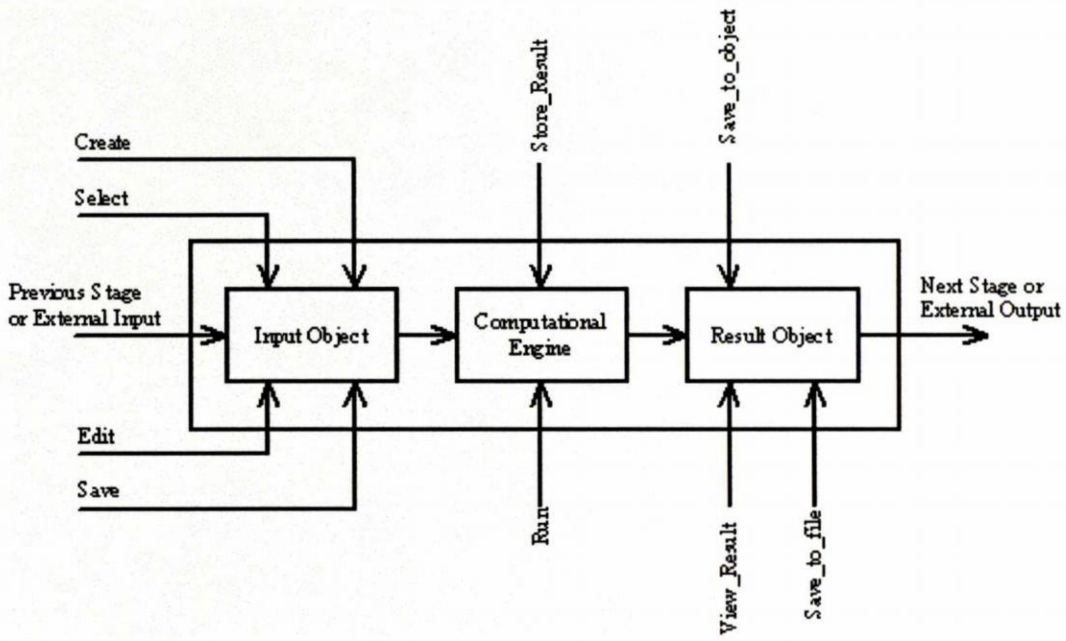


Figure 3.1 Component Architecture

From a mathematical perspective, the input would be a vector. The vector would be placed within the computational engine (model application). Once the analysis is complete, the computational engine produces an output vector. Figure 3.2 illustrates this process.

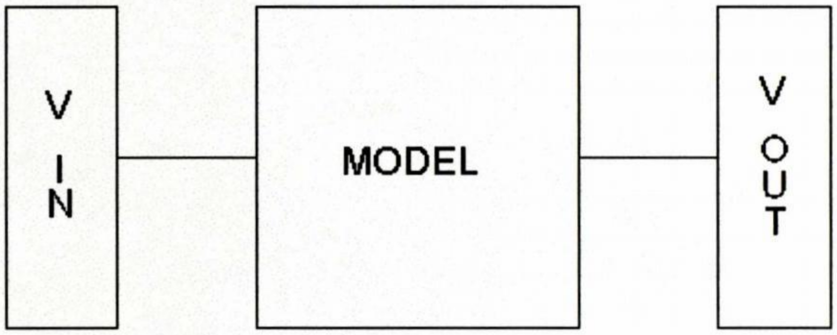


Figure 3.2 Mathematical Model

The heart of the Component Architecture is the Computational Engine. The Computational Engine will be unique for each component, and the cascading of the individual components will yield an overall Systems Model. Figure 3.3 illustrates the System Model.

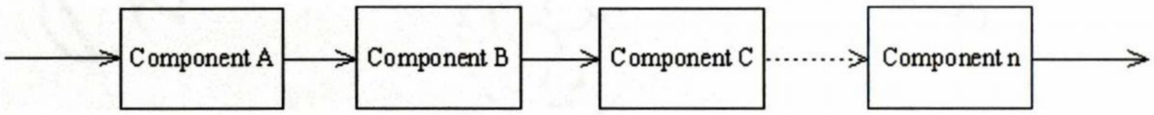


Figure 3.3 Conceptual Compositional Model

3.3.2 Operating Systems Model

The Operating System Model has characteristics of a typical operating system. The Operating System architecture is illustrated in Figure 3.4.

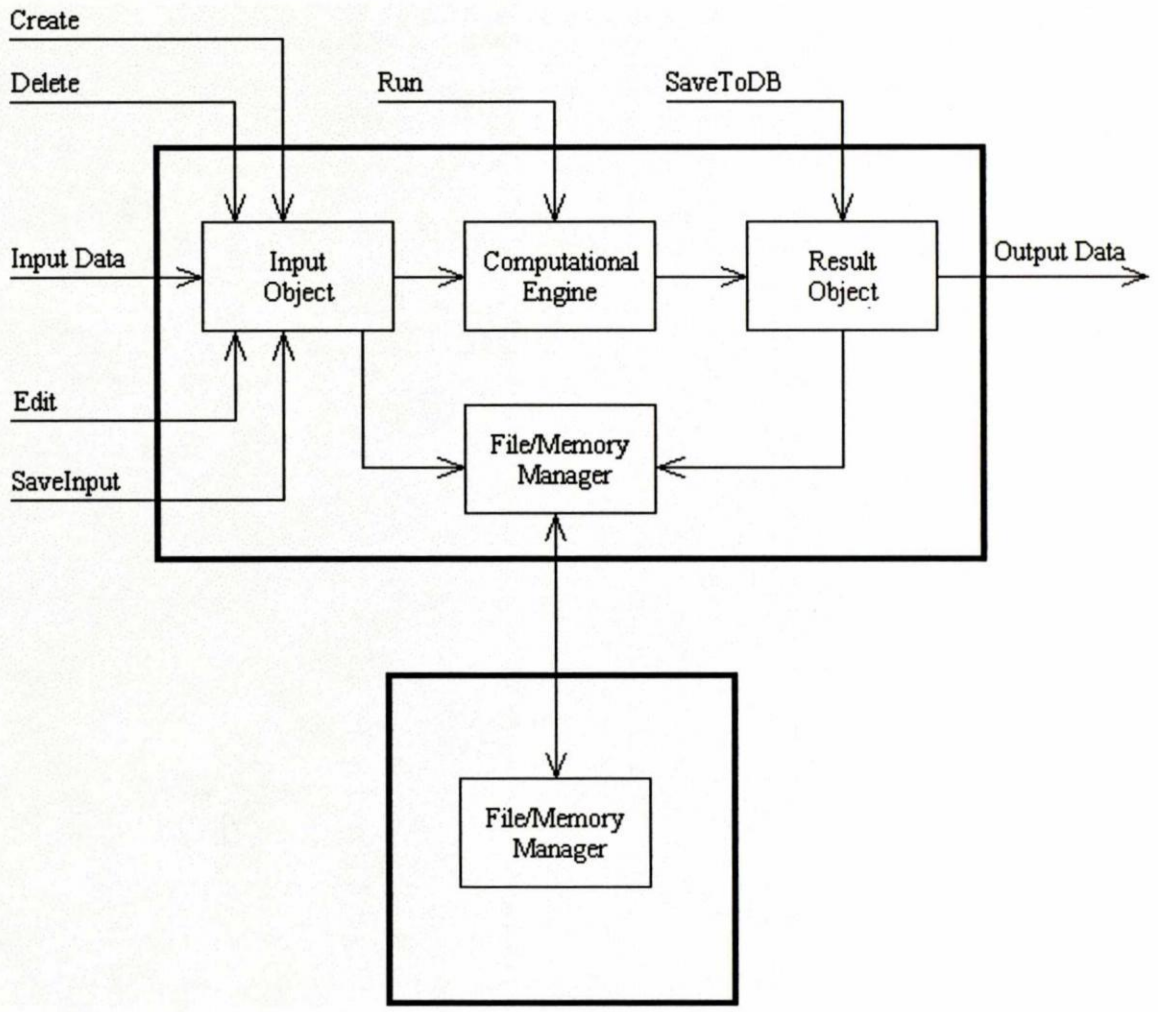


Figure 3.4 Operating System/Application Architecture

The operating system architecture resembles that of the component architecture in Figure 3.1; however, in this architecture the application monitors file/memory usage and communicates that information to the operating system. The operating system would know the allotted amount of available memory and would prevent further memory usage if there were insufficient memory.

From the operating system architecture the conceptual operating system model was developed and is illustrated in Figure 3.5. In this model, it is shown that the operating system monitors file and memory usage from all applications, where each application is providing the information to the operating system. Note that the operating system does not provide any data input or result output. In this model, that responsibility falls upon the application.

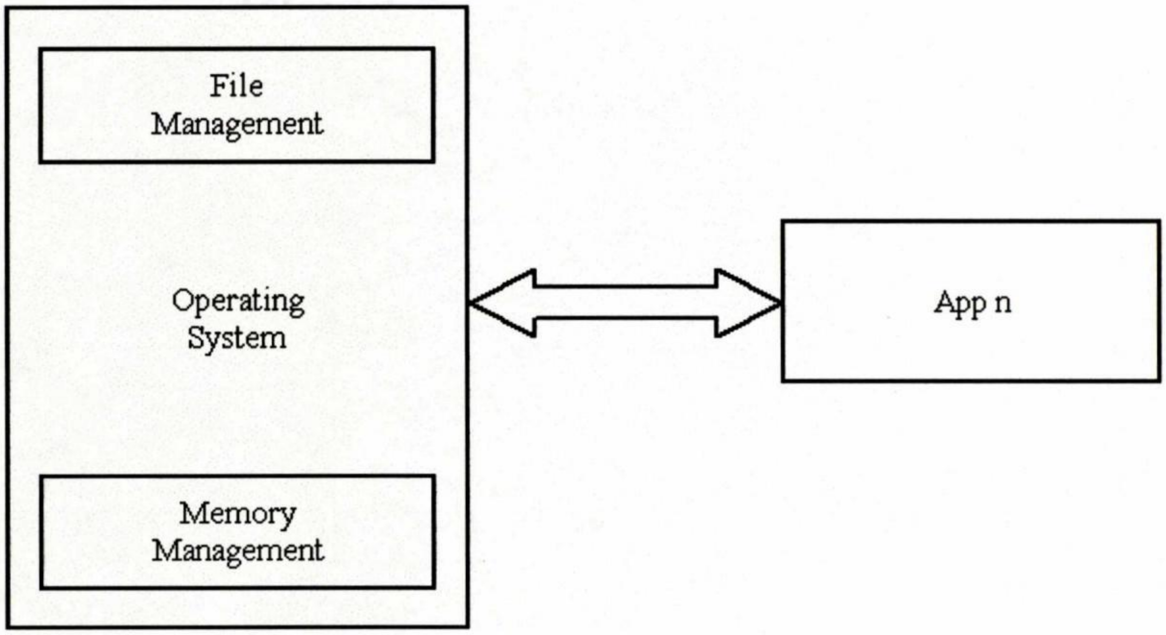


Figure 3.5 Conceptual Operating System Model

3.3.3 Shell Model

In the Shell Model, all applications are installed within an environment known as a shell. The architecture for this model is shown in Figure 3.6. Within the architecture is the computational engine. Input to the computational engine can come from one of three sources. Those sources are from a new input object, an existing input object, or from a converted result object. The new input object is created from an external database, and is defined as never being used within the computational engine. The existing input object is defined as an object that has been applied to the computational engine at least one time. The converted result object is in reality an input object.

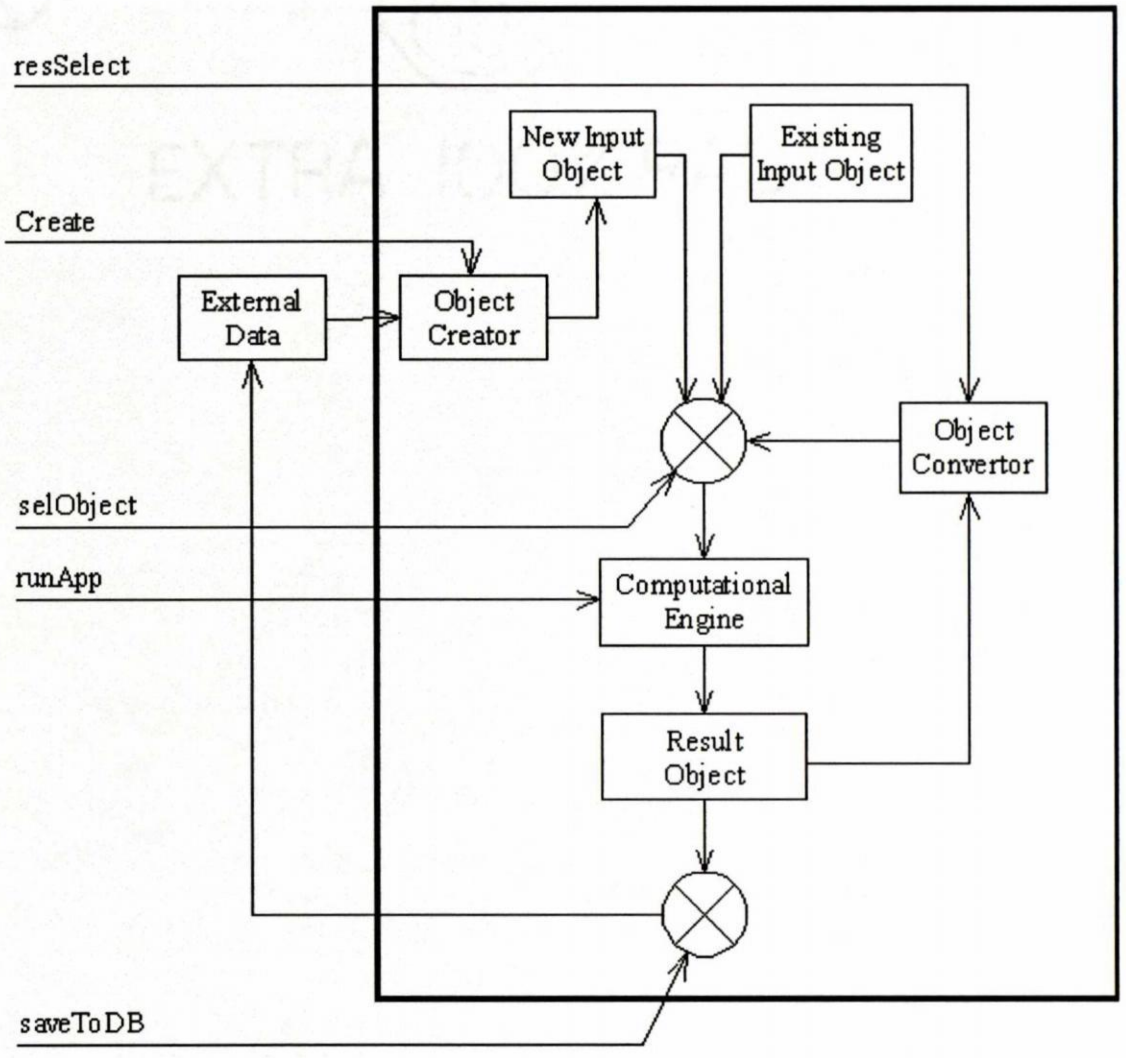


Figure 3.6 Shell Architecture

The various control inputs are used to create input objects, select a specific object as an input to the computational engine, run the computational engine, and save the results to the external database.

The conceptual model of the shell is illustrated in Figure 3.7. Any number of applications can be run from within the shell, limited only by physical constraints, such as hard drive space. As with the architecture, there are three possible inputs to any application. The inputs are from an external source, an existing input, or from previous results.

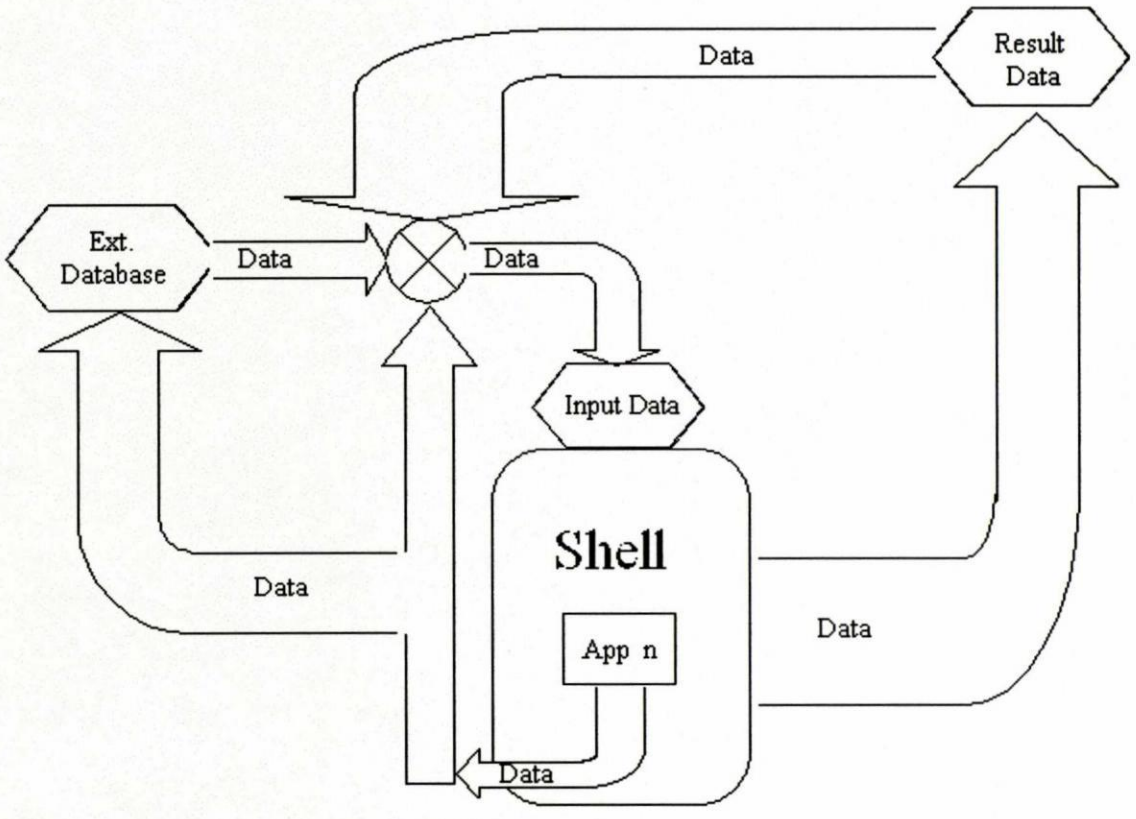


Figure 3.7 Conceptual Shell Model

3.3.4 The GPET Model

In §3.1, it was stated that the conceptual models that represents GPET, in fact, contain characteristics of the Compositional, Operating System, and Shell models, with the major emphasis on the Shell model. This section discusses how, out of the three separate models, the GPET Model evolved.

In the Compositional Model, individual components are cascaded to ultimately develop a compositional model. In GPET, several model applications can be cascaded, ultimately creating a compositional model.

In the Operating Systems model, File and Memory Management mechanisms are incorporated. In GPET, File and Memory Management mechanisms have been implemented to conserve memory and file space when adding and deleting experiments.

In the Shell Model, applications are installed within an environment where the data is contained within objects, and resources are provided for evaluation purposes, thus reducing the model application to a computational engine. The GPET Model most closely follows the Shell Model.

In many higher level programming languages software is developed to a particular platform (e.g. Windows). For example, Microsoft Word® is an application that is developed for the Windows® platform. Conceptually, GPET is no different. In order for model applications to function properly within GPET, the application must follow certain rules or it will not work correctly. Consequently, the model application is conceptually being designed for a particular platform, in this case GPET.

3.4 Software System Architecture

Once the interfaces have been identified through the software development stages, software system architecture can be developed. Figure 3.8 illustrates the architecture of the Generic Performance Evaluation Tool (GPET).

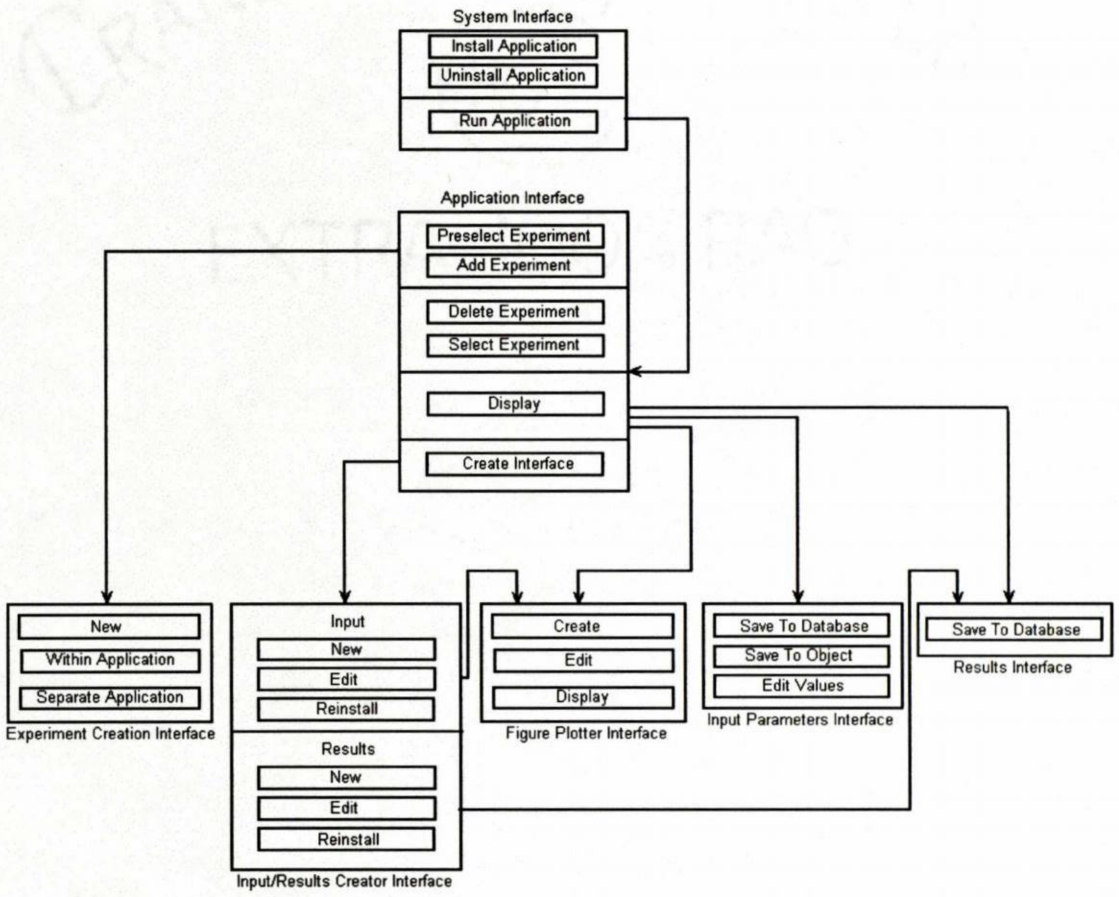


Figure 3.8 GPET Architecture

The following modules are identified in the architecture. They are: Systems Interface, Application Interface, Experiment Creation Interface, Input/Results Creator Interface, Figure Plotter Interface, Input Parameters Interface, and the Results Interface.

This system is a three-tier system. The Systems Interface is the main interface. This interface is at the top of the tier (Top-level). Below the Systems Interface are all Application Interfaces. These interfaces are referred to as Second-level interfaces. Third-level interface include the Input/Results Creator, Experiment Creator, Input Parameter, Results, and Figure Plotter Interfaces.

3.5 Chapter Summary

This section focused on the developmental process involved in designing the GPET software. The chapter begins with an overview of the Michelson Interferometer. Specifically, the components as well as a high-level description of the interferometer, was discussed. Next, various conceptual models were looked at, and from those conceptual models, a hybrid model evolved. Then, the software development stages that were followed, was discussed. Finally, architecture of the tool was created based on the software development process that was followed. The architecture will be the foundation on the implementation of the GPET tool, which will be discussed in detail in Chapter IV.

CHAPTER IV

GPET IMPLEMENTATION

4.1 Introduction

This section will focus on the implementation of GPET. In this section we outline the implementation of GPET in MATLAB. We will begin by describing how objects are developed (§4.2) in our platform of choice and how they function in the environment in which we are working. In order for GPET to work properly with applications, the Application Template is required to work within the shell, and is discussed in §4.3. The main menu in GPET is referred to as the System Interface, and will be discussed in §4.4. From the System Interface any number of applications can be launched. In §4.5 the Application Interface and all associated interfaces are detailed. The chapter concludes with the Chapter Summary in §4.6.

4.2 Object Development

The Problem Statement in §3.4.1 states that the use of input and result data objects will be used to contain their respective data values. In MATLAB, classes are used, along with inheritance. Classes are used to provide encapsulation of data, where attributes contained within one object will be protected (or hidden from) other objects. Classes in MATLAB will be discussed in §4.2.1. Inheritance creates a parent/child relationship between classes, and is discussed in §4.2.2. Figure 4.1 illustrates the application architecture and how objects fit within the application.

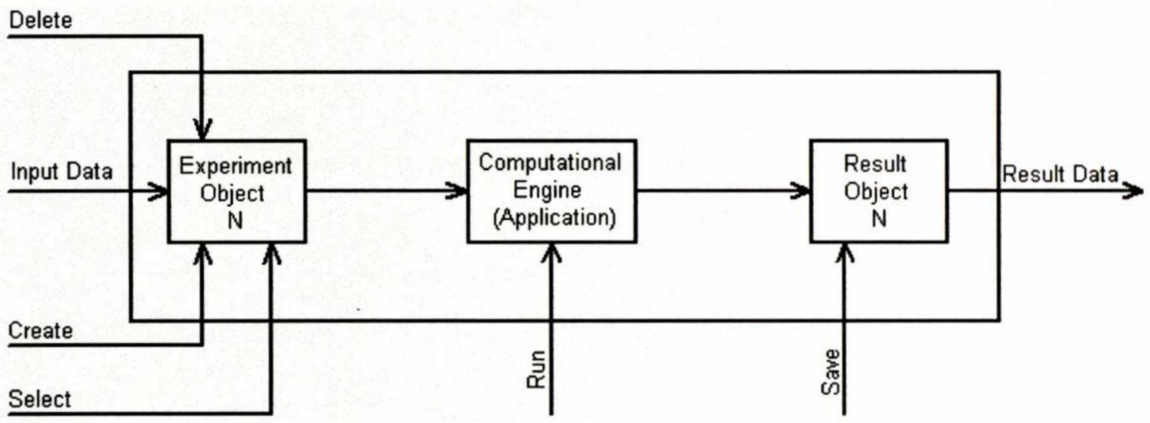


Figure 4.1 Application Architecture

4.2.1 Classes

The MATLAB platform was selected for the implementation of GPET as a result of end users requirement. Objects that are developed in MATLAB mostly parallel objects in higher-level languages such as C++. Chapter 2 (Background) discusses the manner in which MATLAB implements classes in the creation of objects.

GPET makes use of two specific objects, referred to as the Experiment Object and the Result Object. Each of these objects contains a maximum of 200 values that can be used when performing experiments and for saving the results of those experiments respectively. To preserve the data contained within these objects, methods have been developed to perform such tasks as reading, writing, and viewing the data contained within either the Experiment or Results object.

The flexibility of class constructors illustrates another advantage of using classes. A particular class may implement more than one class constructor, which is based on the number of input arguments when creating the class. GPET implements three class constructors for the Experiment Object and two constructors for the Results Object, which will be discussed in more detail in the Experiment Selection Interface section (§4.4.4.1).

4.2.2 Inheritance

When working with inheritance, we are dealing with child and parent classes. GPET makes use of inheritance. With inheritance, a child object inherits all the attributes from its parent object, and is also allowed to call the methods of the parent class. In addition, the parent class is allowed to access those fields that were previously inherited from the parent class, but can not access attribute that are new to the child class.

Whenever a user creates a new experiment, GPET instantiates an Experiment/Results pair. Table 4.1 illustrates the Parent/Child relationship of both the Experiment and Result Objects. For input objects, the child class is the Experiment class and the parent class is the ExpIn class. For result objects, the child class is the Results class and the parent class is the ExpOut class.

Table 4.1 Parent/Child Class Relationship

Parent Class	Child Class
ExpIn	Experiment
Expout	Results

The parent class in each object contains only the values. The child class stores the name that is given to each Experiment/Results object pair. The reason for this is to give a relationship between a particular experiment and the results created from that experiment.

4.3 Application Template

Any application that is run with GPET will function only as a computational engine. In other words, the only purpose of the application is to produce calculations based on specific input values from Experiment Objects. No other functionality should be incorporated into the application. To this end, the development of the Application Template was required. The purpose of the template is to give the procedure for developing applications to run within GPET.

The Application Template is a sequence of eight steps necessary to create the application. Figure 4.2 illustrates a flow diagram of the Application Template.

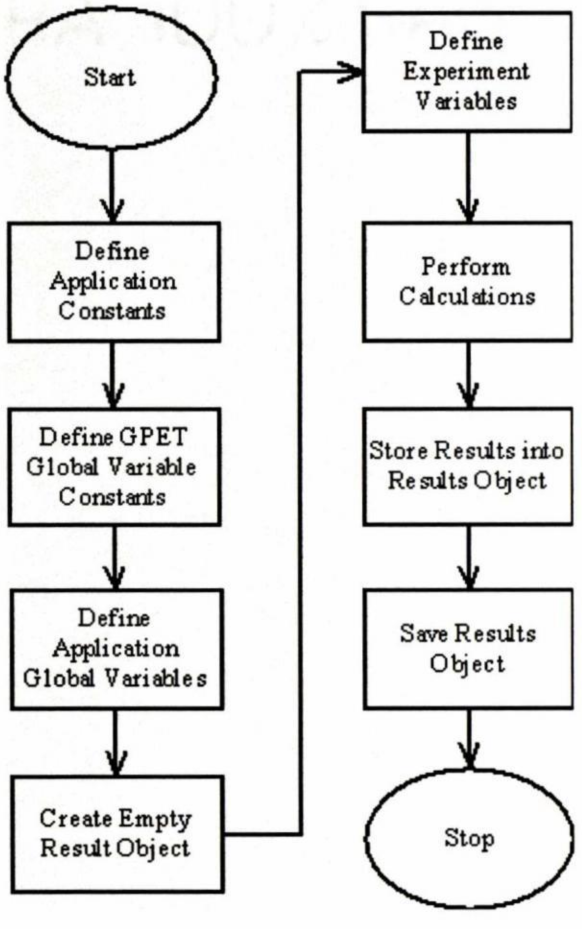

Start
Define Application Constants
Define GPET Global Variable Constants
Define Application Global Variables
Create Empty Result Object
Define Experiment Variables
Perform Calculations
Store Results into Results Object
Save Results Object
Stop

Figure 4.2 Application Template Flow Diagram

The instructions within the template that precede each step must be adhered to for the application to operate properly within the shell. The application template contains the following eight steps. The first step defines any constants that the application requires to function. The second step defines global variables that are required by the GPET software. It is noted within the Application Template that these global variables must not be deleted. The third step contains the global variables that are used to store calculated internal application results after computation. In the fourth step, the application creates a results object that will be used to store the user-defined result values after all computations are performed within the application. In the fifth step, all experiment variables that are used within the application are defined. The sixth step is where all computations within the application are performed. The seventh step stores all user-defined result values into the results object. In the eighth and final step, the result object is saved.

4.4 System Interface

The System Interface is the main interface for the GPET-GIFTS system. The interface provides the functionality to install, uninstall, and launch applications. Figure 4.3 illustrates the System Interface. In this section, the implementation of how applications are installed and removed from the system, are discussed.

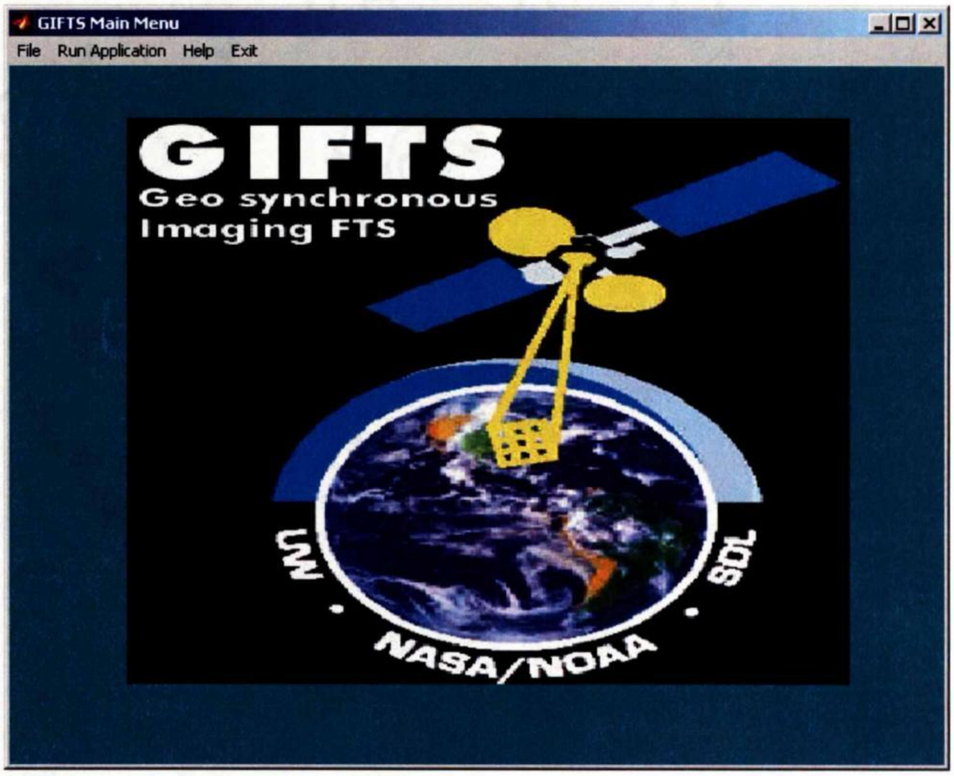


Figure 4.3 GPET-GIFTS Main Menu Interface

4.4.1 Installation of Model Applications

The underlying code that supports the installation of applications performs three major tasks. First, the application name that is provided will be applied to the Systems Interface to give the ability to launch the application. Second, a MAT file will be created to hold all information that the application will require to function properly. Third, an application interface will be created that will be used to provide all the necessary resources to run experiments, as well as view the results of performed experiments.

The first step for installing an application is to provide a unique application name. GPET verifies that the application name is unique and legal. Each application resides in its own directory and GPET updates the MATLAB search path to make the application available.

Next, the System Interface is updated to make the new application available through the GUI ‘Run Application’ menu. Furthermore, relevant pull-down menus are associated with the new application in the Application Interface to be discussed in more detail in §4.5

In order to keep information about one application separate from other applications, part of the installation process creates an application MAT file. This file stores 12 different pieces of information relating to the application. These pieces of information include such data as the number of experiments associated with the application, the names of those experiments, the actual experiment and result objects, along with other pertinent data. Table 4.2 summarizes the 12 MAT file variables that each application uses.

Table 4.2 Application MAT File Variables

MAT Variable	**Description**
ExpNameCell	Cell Array to store experiment names
ExpObjCell	Cell Array to store Experiment objects
MATFile	MAT file name
MAX_COUNT	Maximum number of experiments installed
ResObjCell	Cell Array to store Result objects
appINITFile	Variable Template file name
AppPath	Array that contains full directory path to application
expr_counter	Number of experiments installed
inputWindowCounter	Number of Input Interface windows created
resultWindowCounter	Number of Result Interface windows created
trackCounter	Number of experiments that have been deleted
trackDelExp	Cell Array to store deleted application names

4.4.2 Removal of Model Applications

Complementing application installation, the System Interface also provides the ability to uninstall applications. A flow diagram of the uninstall function is provided in Figure 4.4.

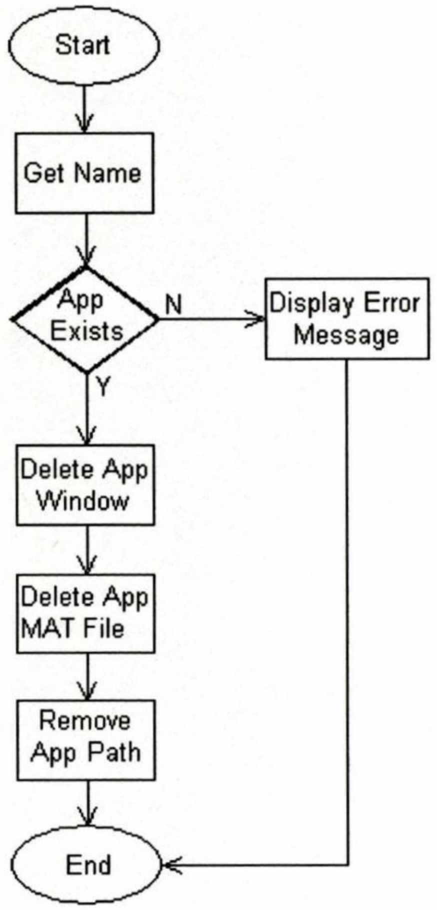


Figure 4.4 Uninstall Flow Diagram

This function performs four tasks. First, the user is prompted for the name of the application to remove. Second, the name is checked against currently installed applications to ensure that the application exists. If the application does not exist, the user will receive a message stating such. Assuming the application does exist, the next step removes the previously create application window. Third, the application MAT file is deleted. Fourth and finally, the application path is removed using the MATLAB Path Browser.

4.5 Application Interface

Figure 4.5 illustrates a typical Application Interface. Once the interface is created, it will serve a multitude of functions, including adding and deleting experiments from the application, as well as selecting individual experiments to use within an application. The interface will also provide the creation of customized Input and Result Interfaces for viewing data in a textual format, and provide the functionality of creating various types of graphical plots. The remaining part of this section is devoted to the implementation of what has just been described.

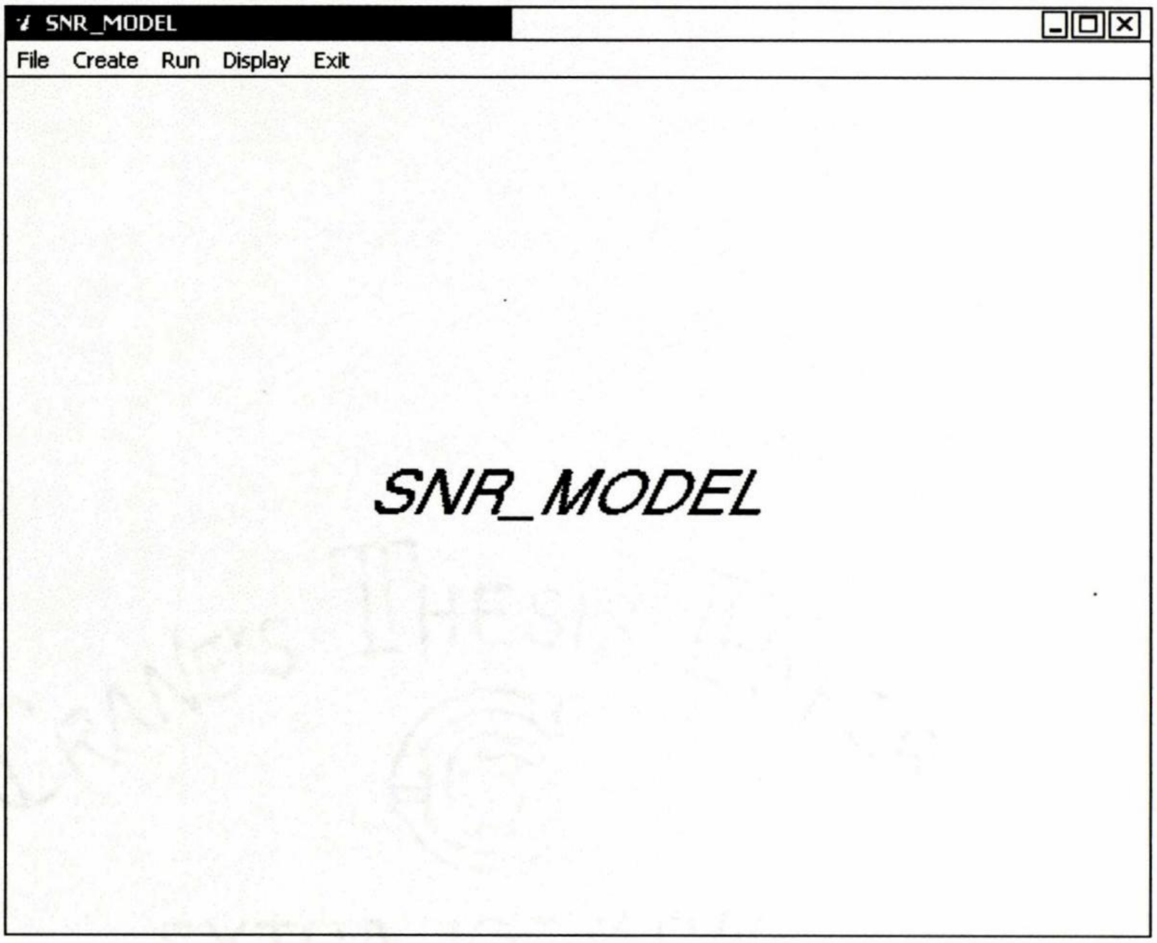


Figure 4.5 Typical Application Interface

4.5.1 Adding Experiments

The functionality necessary to add experiments is encapsulated in the 'AddExp' function, which is called through the Experiment Selection Interface.

Experiments may be created in one of three ways. First, default values for an existing application can be read from a database. Second, values can be taken from an existing experiment object within the same application. Third, values from a Result Object from a different application. The overall process is illustrated in Figure 4.6.

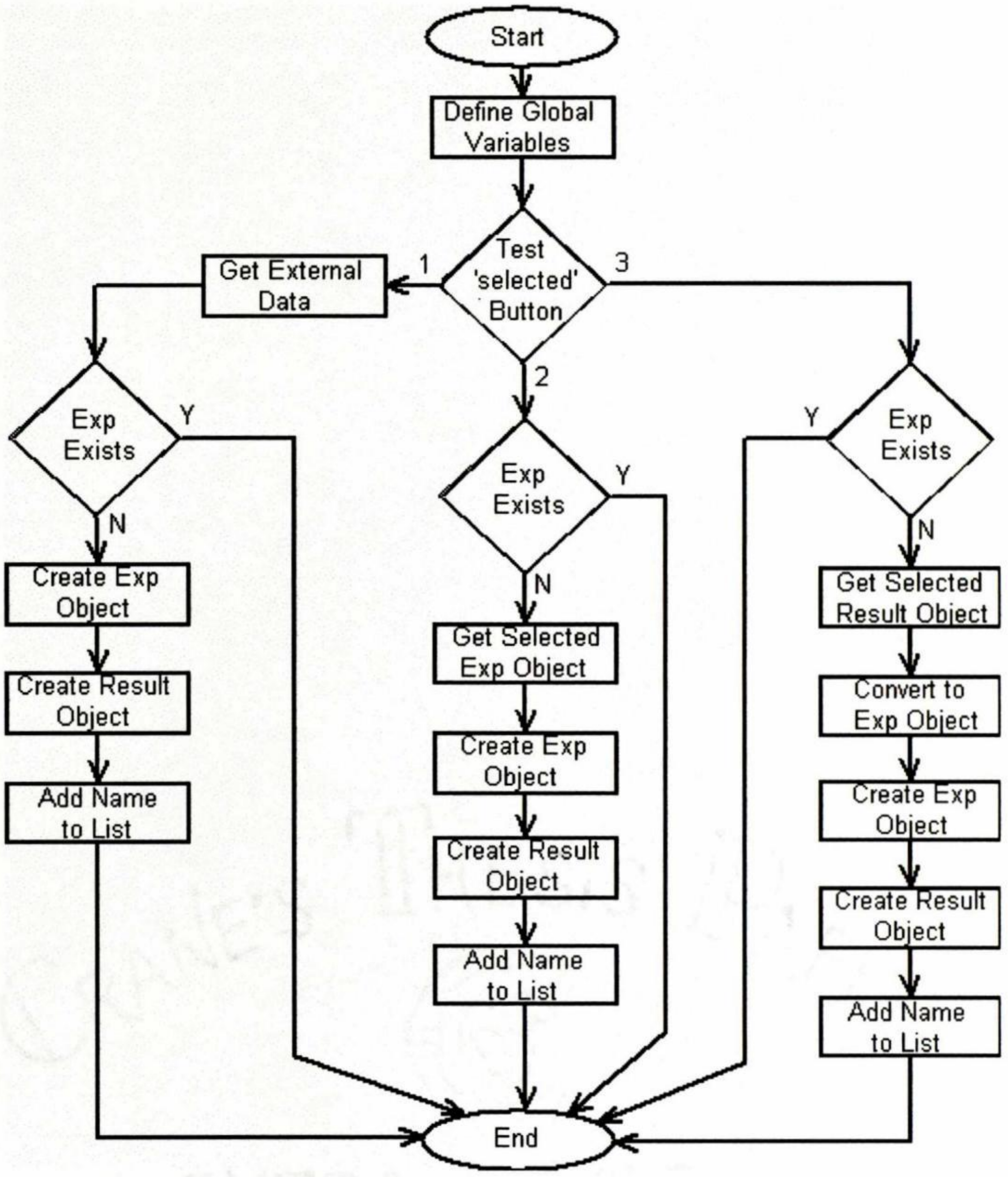


Figure 4.6 'AddExp' Basic Flow Diagram

4.5.1.1 Experiment Selection Interface

The Application Interface still exists when the Experiment Selection Interface is selected. To prevent inadvertent software functionality, certain selections within the Application Interface are disabled, in the event the user brings the Application Interface

Window into focus. There are a number of functions associated with the Experiment Selection Interface shown in Figure 4.7.

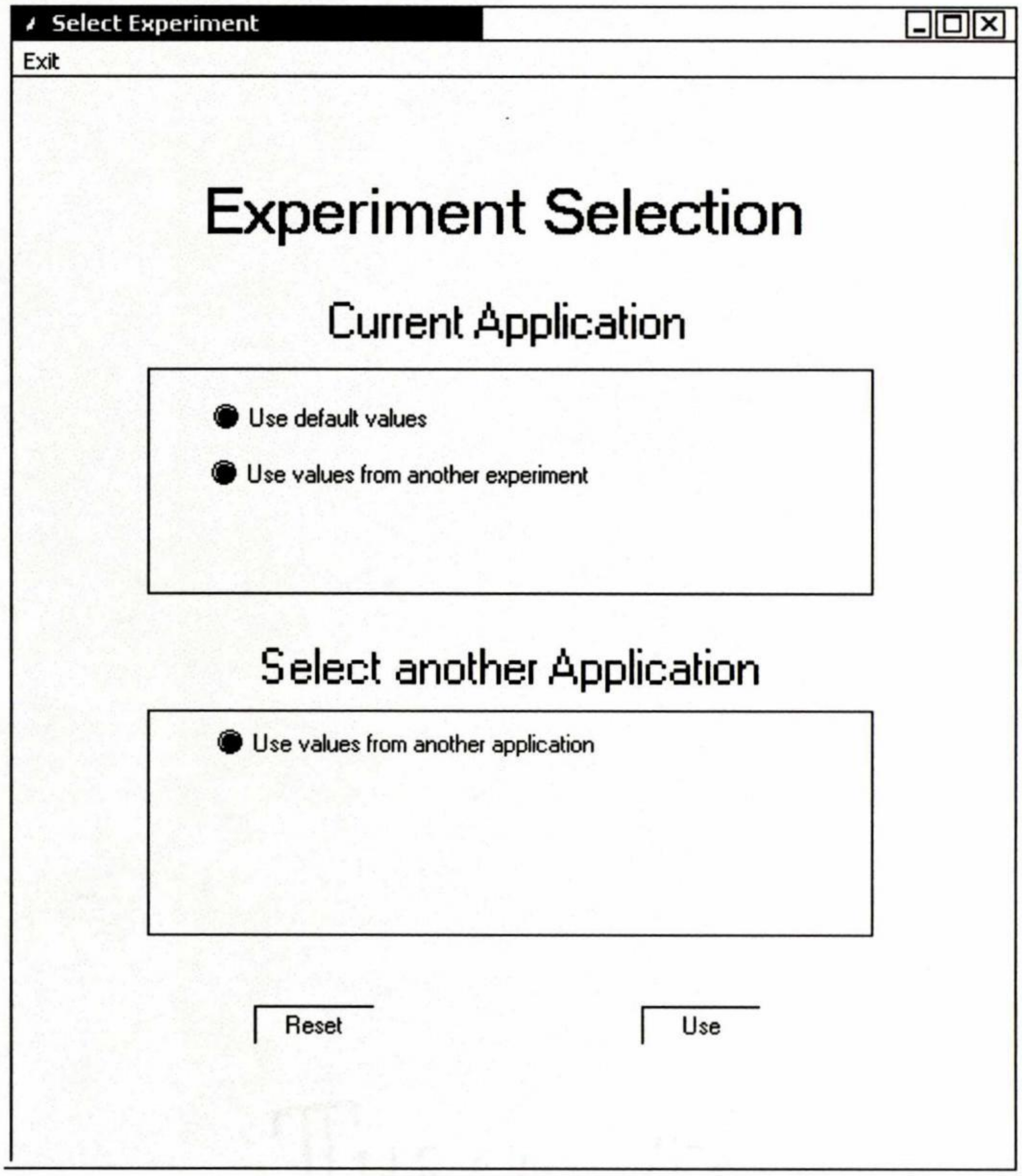

Select Experiment
Exit
Experiment Selection
Current Application
Use default values
Use values from another experiment
Select another Application
Use values from another application
Reset
Use

Figure 4.7 Experiment Selection Interface

Once the Experiment Selection Interface is in focus there are three radio buttons to select from and two pushbuttons. When the top radio button is selected, default values from a user-generated database are used. When the second radio button is selected, values are used from a previously created experiment within the same application.

Situations arise when the user wants to use the results of one application as input to another application. Option three was developed to support this capability. Recall from the installation section that when an application is installed, an application MAT file is created, containing all information required by a particular application. The source application MAT file is temporarily loaded into the user workspace. The user is then given a list of the experiments to choose from, and its respective result object data will be used as input to the destination application. Then the destination application is then reloaded. Figure 4.8 illustrates option three.

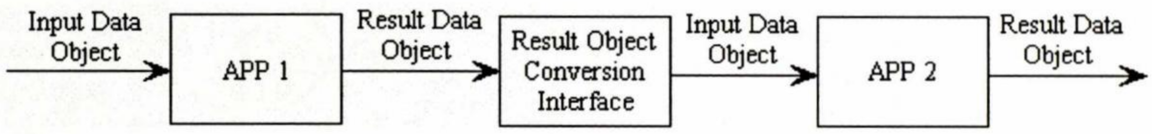


Figure 4.8 Experiment Selection Interface Option Three

The first of the two pushbuttons is the ‘Reset’ button. The action in response to clicking on this button clears all the radio button selections and hides any visible list boxes. It also checks to ensure that if a different application MAT file is loaded into the user workspace, it is reset to use the currently open applications MAT file.

The other pushbutton is the USE button. The actions in response to clicking on this button includes storing the appropriate information that will be used in the second part of experiment addition process, resets the user workspace back to the currently running application if needed, then closes the Experiment Selection Interface.

Once the appropriate selections have been made to create a new experiment, interlocks are used to prohibit certain selections within the Application Interface until the experiment has been completely added. Table 4.3 lists the various functions used with the Experiment Selection Interface, along with details of the purpose of the function.

Table 4.3 Experiment Selection Interface Functions

Function	Purpose
exprSelection_Create	Disables certain selections within the Application interface in the event the user brings the Application Interface into focus.
defaultValues_callback	Allows the user to select only one of the options at a time. Sets the 'selectButton' variable to '1'.
ExpSelradio2_callback	Allows the user to select only one of the options at a time. Sets the 'selectButton' variable to '2'.
ExpSelradio3_callback	Allows the user to select only one of the options at a time. Sets the 'selectButton' variable to '3'.
AddExpSel	Loads the selected source application MAT file.
ExpSelreset_callback	Clears all radiobutton selections. Hides any visible list boxes. Resets application MAT file to currently open application.
UseButton_callback	Stores appropriate data that will be used in the currently running application. Resets user workspace to currently running application. Closes Experiment Selection Interface.
exprSelection_DeleteFcn	Enforces interlocks to guarantee proper operation for sequential user input.

4.5.1.2 Object Instantiation

As previously discussed, GPET provides three class constructors to instantiate Experiment Objects and two class constructors to instantiate Result Objects. For Experiment Objects, the first constructor sets all values to zero, the second constructor uses a previously created object, and the third constructor is used when creating a new object from an existing external database (text file). Currently, GPET uses the second and third constructor, leaving the first constructor for future development. A partial listing of the pseudo code for the constructors of the Experiment class is shown in Figure 4.9.

```
function ei = Experiment (name, varargin)
switch nargin
case 1
  Set object name field to input argument name
  While n is less than 201
      Set cell array n to 0
  Call ExpIn function using cell array as input argument
  Create experiment object
case 2
  Set object name field to input argument name
  Set internal experiment object equal to variable input argument one
  Convert internal experiment object from numeric values to a cell array
  Create experiment object
case 201
  Set object name field to input argument name
  Create internal cell array from input arguments
  Call ExpIn function using cell array as input argument
  Create experiment object
Otherwise
  Output error message due to invalid number of input arguments
end

function expinput = ExpIn (values)
While n is less than 201
    Set experiment field value n to cell array value n
.Create internal experiment object
end
```

Figure 4.9 Pseudo code for the Experiment class

Initially, the Experiment function is called (ultimately an experiment object will be created). The Experiment class is the child class, and will call upon the ExpIn function (parent class) to apply the 'varargin' (if available) values to the fields within the ExpIn class, and will create an ‘expin’ object. When the ExpIn function ends, the object is used as an input argument when creating the experiment object. In all cases, objects must be given a name.

Case 1 is used to create an object, where all 200 values within the object are set to zero (this case is not used in the current version of GPET). The case statement sets all the values to zero, so there is no need to have a ‘varargin’ variable, and the ‘nargin’ (number of arguments) value is equal to one (all objects must have a name). After all values have been set to zero, the ExpIn class (parent class from Table 4.1) is called to create an ‘expin’ object.

Case 2 is used when option two or three is selected on the Experiment Selection Interface. In both of these cases, objects already exist, either from an existing experiment within the same application, or from a converted Result object from a different application. In either of these situations, an internal object is created, and applied as an input argument when instantiating the new experiment object. In case 2, the ExpIn parent class is not used.

Case 201 is used when option one is selected on the Experiment Selection Interface. With this option we are using an external database, and therefore must assign each of the 200 input values individually. The ‘varargin’ argument will contain 200 arguments. The switch argument ‘nargin’ will be equal to 201 (the name plus the 200 input arguments). In Case 201, a cell array is created from the ‘varargin’ variable. The cell array is then used as an input argument to the ExpIn class when it is called. The ExpIn class will create an ‘expin’ object, and that object will be used as an input argument when instantiating the experiment object.

Instantiation of a Results Object is similar to that of an Experiment Object with the exception that there are only two class constructors instead of three. Figure 4.10 illustrates the class constructors for the Result (child) and ExpOut (parent) classes.

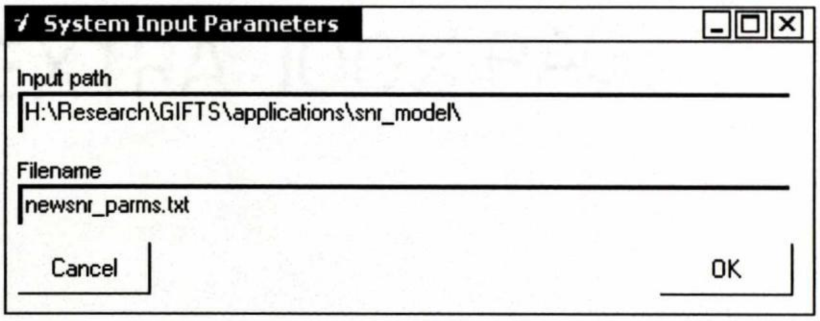
```
function ER = Results (name, varargin)
switch nargin
case 1
  Set object name field to input argument name
  While n is less than 201
     Set cell array value n to 0
  Call ExpOut function using cell array as input argument
  Create Result object
case 201
  Set object name field to input argument name
  Create internal cell array from input from input arguments
  Call ExpOut function using cell array as input argument
  Create Result object
Otherwise
  Output error message due to invalid number of input arguments
end

function ExpResult = ExpOut (values)
 While n is less than 201
    Set Result field value n to cell arrya value n
 Create internal Result object
 end
```

Figure 4.10 Pseudo Code for Result class

4.5.1.3 Adding the Experiment

Adding an experiment to the application is a two-step process. The first step involves the use of the Experiment Selection Interface, where the user is given three options to select from in retrieving data. Once a decision is made on which option to use, the 'AddExp' function is called. When 'AddExp' is started, information is collected based on the option that was selected during step one.

Option 1 the Experiment Selection Interface (Use Default Values) first presents a dialog box. The dialog box shown in Figure 4.11 asks for the input path and filename of the database in which to retrieve data. Since MATLAB is matrix-based, the database must be implemented as a matrix as well.



Input path

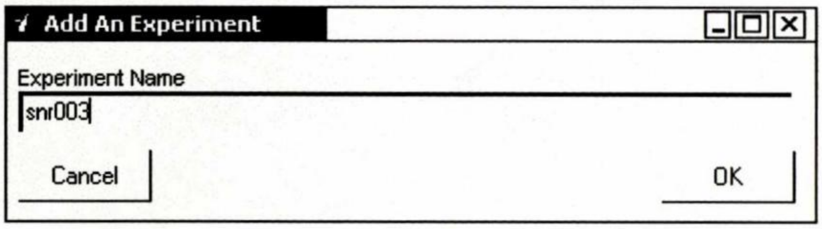
H:\Research\GIFTS\applications\snr_model\

Filename

newsnr_parms.txt

Figure 4.11 Directory and Filename Dialog box

The database file (text file) is configured as a 20x20 matrix so that both experiment and result values can be contained in one database file. The constraint comes from user consensus of having 200 input and 200 result variables within the respective objects. The overall yield is 400 variables, and thus would fit within the 20x20 matrix. The first 20 rows by 10 columns are reserved for experiment values. The remaining 20x10 section is reserved for results that the user may wish to save. The values from the database are loaded into the user workspace in a user-defined variable. Once the data is inserted into the database, another dialog box (Figure 4.12) appears asking the user to provide a unique experiment name. Assuming the experiment name does not exist, an experiment object is created by applying each value within the user-defined variable as an input argument.


Add An Experiment

Experiment Name

snr003

Figure 4.12 Experiment Name Dialog box

Along with the experiment object a result object is also created. The only input argument to the result object is the name. The name given is the same name as the experiment object.

There are two cell array variables that store all experiment and result objects for a specific application. If there are currently five objects stored in each of the cell arrays, and the third object pair is to be deleted, there would be an empty cell in the middle of both cell arrays. If there were no way of keeping track of empty cells, memory would be wasted. Therefore, the next step in adding an experiment/result object pair is to identify any empty cells, and if one exists, use that cell to store the currently created experiment/result pair. If no empty cells exist, then add the experiment/result object pair at the end of their respective cell arrays.

The next step enables and disables certain selections on the Application Menu. Figure 4.13 illustrates the specific areas that are enabled and disabled.

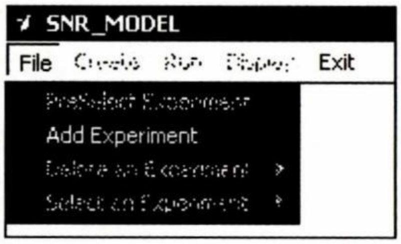
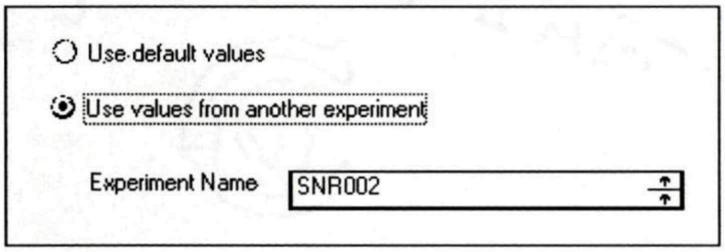


Figure 4.13 Disabled Selections

There is only one selection enabled, and that selection is 'Add Experiment'. The final step for option 1 is to update the running application MAT file. This is the file that contains all information pertinent to a particular application.

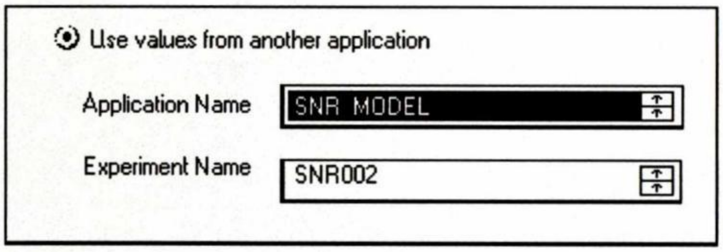
Option 2 on the Experiment Selection Interface (Use values from another experiment) is similar to option 1 with a few exceptions. In option 2, an external database is not used for retrieving data, so there is no need to ask for an input path and filename. The data is already contained within another experiment object in the same application. Figure 4.14 illustrates the selection of option 2. The remaining steps of tracking deleted experiment/result object pairs, menu enabling/disabling, and updating the application MAT file is performed in the same manner as option 1.



Experiment Name: SNR002

Figure 4.14 Option 2 Selection

For the most part, option 3 on the Experiment Selection Interface (Use values from another application) creates an experiment/results pair much like that of option 2. For option 3 however, we are creating an experiment object by using the results of an experiment within a different application. Figure 4.15 shows the option 3 selection. On the surface this may not seem to be a difficult task. The task would include opening the other applications' MAT file, get a copy of the result object, and store that value within a temporary object, then use the temporary object as an input argument when creating the new experiment.


● Use values from another application

Application Name: SNR MODEL

Experiment Name: SNR002

Figure 4.15 Option 3 Selection

However, a result object cannot just be placed into an experiment object. The two objects may look similar, but they are indeed two totally different object types. The result object must be transformed into an experiment object before the experiment object can be saved. The 'res2exp' function is used to break down the 200 individual values, and then rebuild those values into an experiment object.

4.5.2 Experiment Removal

The process of removing an experiment begins by selecting 'Delete Experiment' from the File pull-down menu. When this is selected another menu will appear with the names of all current experiments. Figure 4.16 shows an application window with 'Delete Experiment' selected and a list of available experiments to delete.

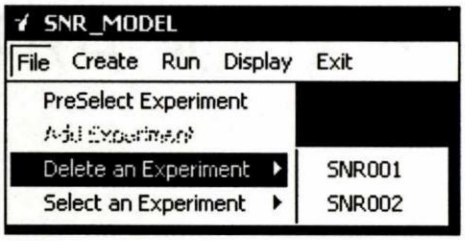


Figure 4.16 Selection for Deleting an Experiment

Once the particular experiment is selected, a search of the experiment cell array is performed to find the particular experiment to be removed. There is an index number associated with each element within the cell array. This index number is also associated with the experiments result object in result cell array. Once the experiment name is found, the index number is used to actually replace the experiment and result objects with an empty matrix. Then, the tracking variables used to note where experiments and results have been deleted are updated in the event that a new experiment is added. Next, the experiment name is removed from the 'Select Experiment' and 'Delete Experiment' pull-down menus within the Application Interface. The last step is to update the application MAT file with the current objects contained within the experiment and result cell array variables.

4.5.3 Selecting Experiments

Under the File menu in the Application Interface a particular experiment is selected to use prior to running the application. Figure 4.17 illustrates the selection of a particular experiment. When a particular experiment is selected, the 'ExpObjCell' cell array variable is searched to find the object by comparing the selected experiment name from the pull-down menu to each object name until the correct object if found.

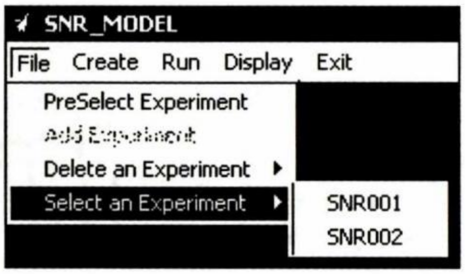


Figure 4.17 Selecting an Experiment

4.5.4 Input/Results Creator Interface

The Input/Results Creator Interface was developed to provide the ability to create input and/or result interfaces that allow the viewing of input and result variables in a textual format. The Input/Results Creator Interface is shown in Figure 4.18. In the case of input variables, the variables can be changed prior to running an experiment with a particular application. In the event that a particular application is uninstalled, and later reinstalled, previously created interfaces can be reinstalled without starting from scratch. The ability exists as well to edit an existing input or result interface. When this selection is made, a list of available input (or result) interface(s) is/are displayed within a list box to choose from. There is a limit on the number of input and result interfaces that are created. With the maximum limit of 200 input and 200 result variables, a maximum of 20 pages of input and 20 pages of result interfaces can be created respectively.

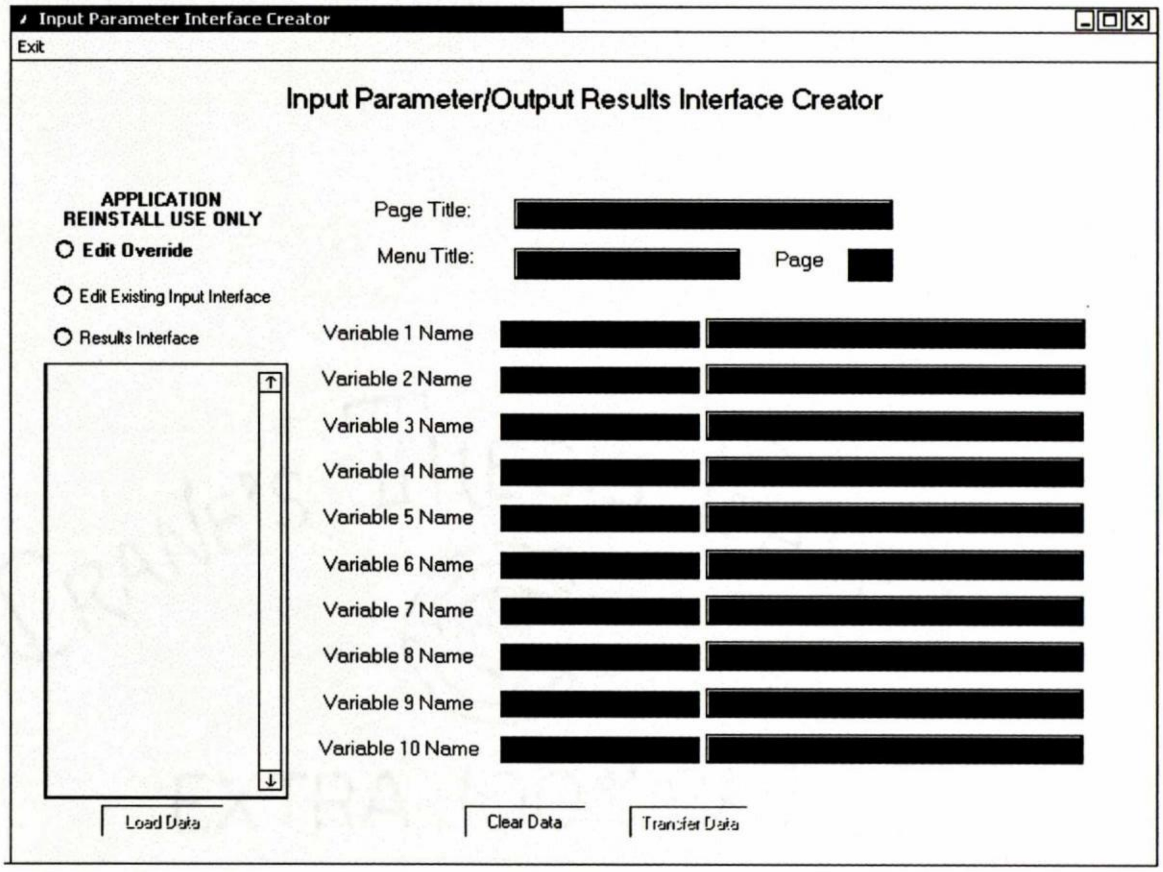


Figure 4.18 Input Parameters/Output Results Interface Creator

In the subsections to follow, discussions will begin with the Variable Template. The template is used in conjunction with the Input/Result Interface Creator Interface to provide internal variable names. The internal variables names are displayed on the Input/Result Interface Creator Interface to aid the user in providing formal names to the internal variables. The subsections following the Variable Template will discuss the creation of an Input Parameter Interface, and a Result Interface, using the Input/Result Interface Creator Interface. The last subsection discusses the ability to reinstall Input and Result Interfaces when an application has been uninstalled, and later reinstalled.

4.5.4.1 Variable Template

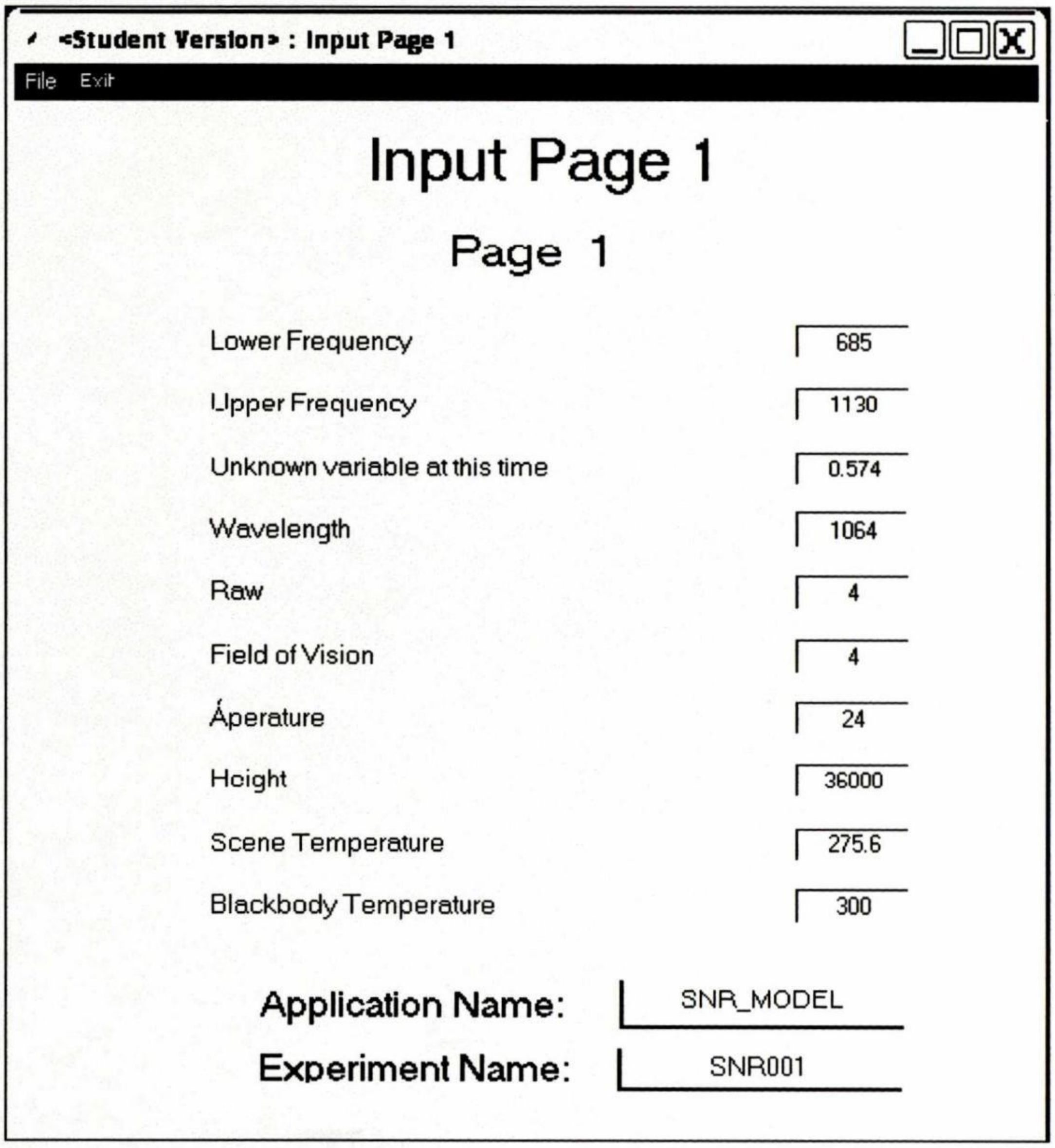
The purpose of the variable template is to identify which of the ten variables on an input or result interface belongs to a specific internal variable within the application. Using this template, the user associates internal variable names with formal names within the application.

The Variable Template is a file that contains MATLAB code, and the user modifies the template by adding internal variables names within the template, that are contained within the users' model application. The variable names entered within the template will be displayed on the Input/Result Interface Creator Interface whenever the user creates input and/or result interfaces, and aids the user in giving formal names to the internal variables. Appendix C contains the Variable Template along with instructions contained within the template on how to set up the template. Additional information on the Variable Template is found in the Users' Manual in Appendix B.

4.5.4.2 Input Parameter Interface

The Input/Result Interface Creator creates an Input Parameters Interface by default. After page title and menu title information are supplied, the Transfer Data pushbutton becomes enabled, allowing the user to associate formal names to their internal variables. These names are retrieved from the variable template that was discussed in §4.5.4.1.

Once the desired associations have been entered, the user commits these associations. The underlying code for the pushbutton event performs several tasks, and those tasks are listed in Table 4.4. One of the tasks performed, but not listed in the table is the manner in which newly created interfaces are named. Figure 4.19 illustrates an Input Interface. The formal names are displayed on the left side of the interface.



Input Page 1

Page 1

Lower Frequency	685
Upper Frequency	1130
Unknown variable at this time	0.574
Wavelength	1064
Raw	4
Field of Vision	4
Aperature	24
Height	36000
Scene Temperature	275.6
Blackbody Temperature	300

Application Name: SNR_MODEL

Experiment Name: SNR001

Figure 4.19 Input Interface

A specific naming convention is applied to both the Input and Result Interfaces. The first three characters are the first three capital letters of the word 'application'. The next character(s) is the application number that has been installed. Following the application number is a character that defines the interface as either an Input (I) or Result (R). The next four characters are the word 'PAGE'. The last character(s) is the page number of either the Input of Result Interface. Once the appropriate filename is created, the files are saved into the same directory in which the specific application is stored.

To clear all entered data on the Input/Result Interface Creator, the ‘Clear Data’ push button is used. This pushbutton performs two tasks. It will clear all information for formal variable names, as well as clear the page name and menu name.

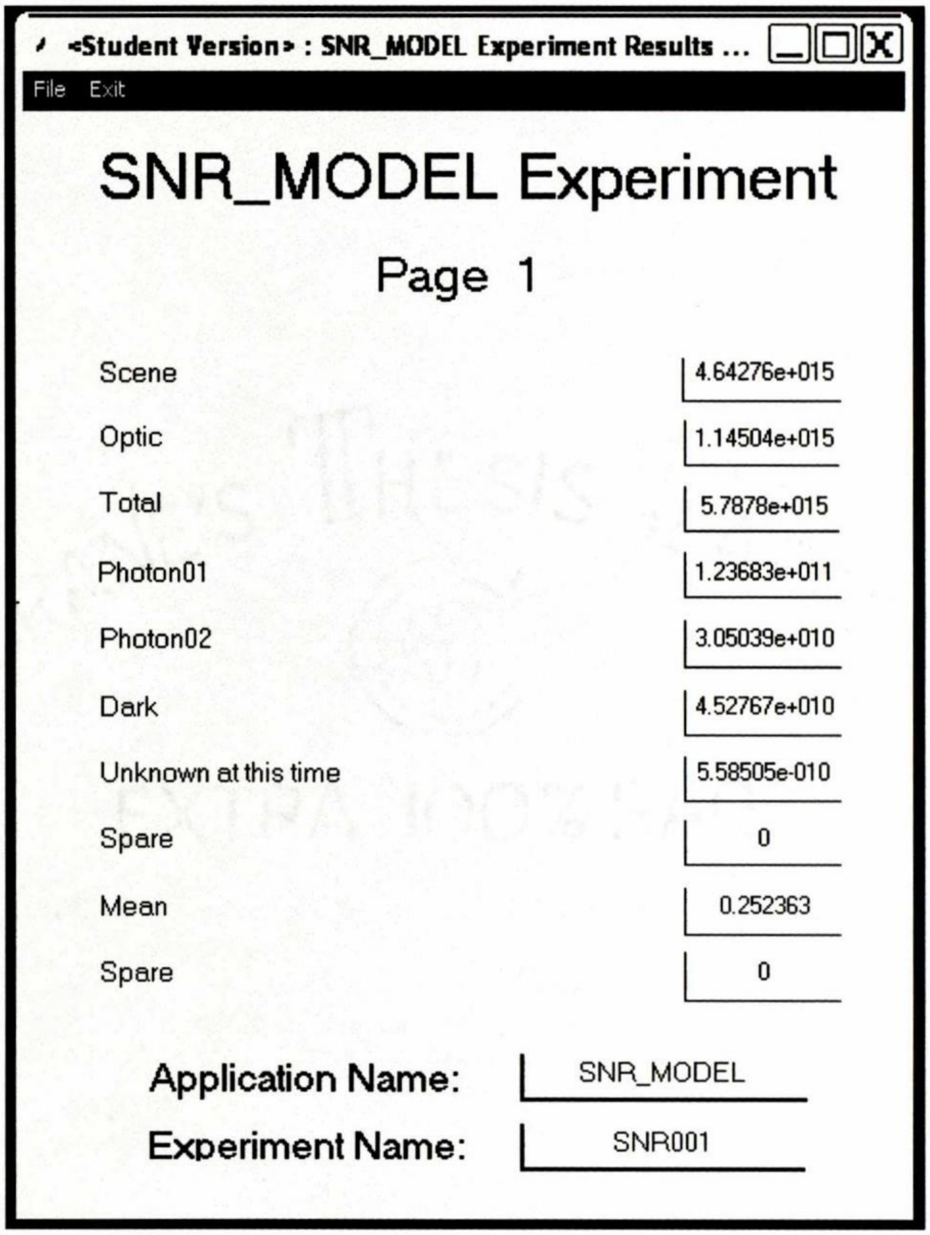
The Input/Result Interface Creator provides the ability to edit existing Input and Result Interfaces. This is a three-step process. First, as a result of selecting the Edit Existing Interface radio button, an event will take place that will display a list box. The list box will list (seeing as we are discussing input interfaces) any Input Interfaces that have been created. Second, clicking on the ‘Load Data’ pushbutton will call an event that will open the specific Input Interface, retrieve formal names, than place those formal names within the Input/Results Interface Creator Interface. Third, when editing is complete, clicking the ‘Transfer Data’ push button calls an event that will send the updated formal names to the currently open interface, as well as saving the edited interface within the currently open application folder. Table 4.4 lists the functions that are implemented within the Input/Result Interface Creator.

Table 4.4 Input/Result Interface Functions

Function	Purpose
Page_menu_callbacks	Prevent interface from creating either an input or result interface until a minimum amount of information is supplied
transfer_data	Tests for maximum number of either Input or Result Interfaces. Transfers data from the Input/Result Interface Creator to the respective created interface
Clr_data	Clears all information for formal variable names, page name, and menu name.
radiobutton1_callback	Update list box when editing an existing Input or Result Interface.
load_data	Retrieves formal names of the interface that requires editing and places them within the Input/Result Interface Creator

4.5.4.3 Result Interface

As was noted previously when the Input/Result Interface Creator is started, it is configured to create an Input Interface. Selecting the Result Interface enables the user to create a result interface. The system is allowed to have up to 20 input pages and up to 20 output pages. Information is entered in a manner similar to the Input Interface. Figure 4.20 illustrates a user-created result interface.



SNR_MODEL Experiment

Page 1

Scene	4.64276e+015
Optic	1.14504e+015
Total	5.7878e+015
Photon01	1.23683e+011
Photon02	3.05039e+010
Dark	4.52767e+010
Unknown at this time	5.58505e-010
Spare	0
Mean	0.252363
Spare	0

Application Name:	SNR_MODEL
Experiment Name:	SNR001

Figure 4.20 Result Interface

Editing a Result Interface is accomplished much like that of the Input Interface with a couple of exceptions. The Results Interface must be selected first to set up for creating a Result Interface. Then, select Edit Existing Interface radio button. A list of all previously created Result Interfaces will appear. The manner of selecting and editing a specific result interface is the same as with an Input Interface.

4.5.4.4 Edit Override

When an application is uninstalled from GPET, the uninstall process does not remove any of the created Input and Result Interfaces for that application from the uninstalled applications directory. If the user later wishes to install the application, it would be somewhat cumbersome to try and recreate the same Input and Result Interfaces. The Edit Override radio button is incorporated within the Input/Result Interface Creator for the purpose of reinstalling existing interfaces.

When the Edit Override radio button is first selected, a method is invoked to warn the user that this button should only be used when reinstalling either Input and/or Result Interfaces after reinstalling an application. 'Edit Override' is used in conjunction with the 'Edit Existing Interface' and the 'Results Interface' radio buttons. A combination of the 'Edit Override' and 'Edit Existing Interface' push buttons will allow a reinstallation of an Input Interface. Selecting all three radio buttons will allow the reinstallation of a Result Interface.

There are a couple of differences between editing an existing interface and reinstalling an existing interface. The first difference is that in a normal edit, all information is available. In a reinstall situation the Menu Title on the Input/Result Interface Creator is missing because of the destruction of the Application Interface when it was uninstalled. The user will need to input Menu Title information prior to being able to allow the transfer of data. The second difference has to do with the counters that keep track of how many Input and Result Interfaces that have been created. In a normal edit we do not want any of the counters to change. We are only doing an edit. However, this is not the case in a reinstall. In a reinstall we are actually "fooling" the system into believing we are creating a new interface even though we are only replacing an already existing one. The checking of all of the radio buttons occurs when we click on the 'Transfer Data' push button.

4.5.5 Figure Plotter Interface

The Figure Plotter Interface gives the added ability to graph the results of currently run experiments. The Figure Plotter has the functionality to create a number of pre-defined plots, allow editing of plots that have already been created, along with displaying a list of all created plots within an application, and displaying a list of all workspace variables for the currently running application. Figure 4.21 illustrates the Figure Plotter Interface.

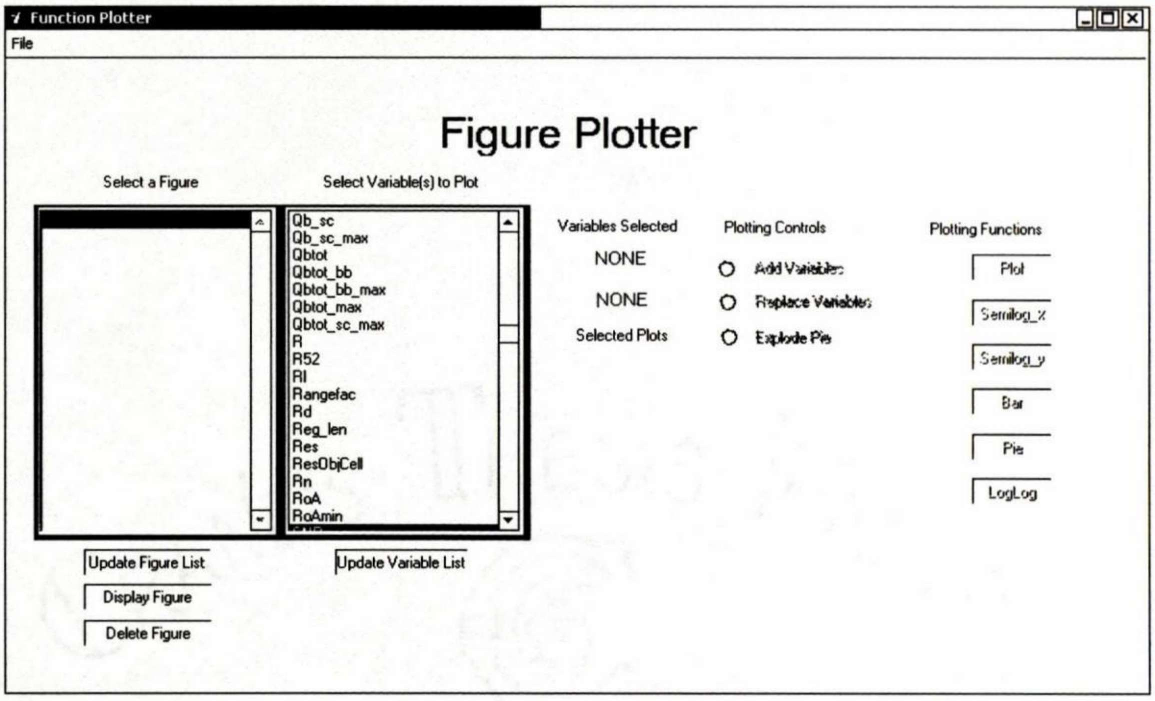


Figure 4.21 Figure Plotter Interface

The Figure Plotter is subdivided into six sections. The 'Select a Figure' section provides a list of previously created plots for an application. The three pushbuttons associated with this section allow the updating of the figure list, displaying a currently selected figure, and deleting an unwanted figure.

The 'Select a Variable to Plot' section lists all variables that are contained in the user workspace, and functionality exists that will update the variable list.

The 'Variables Selected' section displays the currently selected variables that will be plotted. This section is useful when the variable list become lengthy, and provides the user with a quick look at what has been selected. A maximum of two variables can be selected.

The 'Selected Plots' section is an information section. It informs the user of what types of plots that can be selected. The two displays will either be 'Pie Chart' or 'All Other Plots', depending on the number of variables that are selected.

The 'Plotting Controls' section add functionality that will allow adding more variables to an existing plot, replace currently plotted variables, or to 'explode' a pie chart. Exploding is nothing more than separating the largest section of the pie from the remaining part of the pie.

The 'Plotting Functions' section provides functionality to use existing MATLAB plotting functions to create a variety of user-defined plots. Table 4.5 lists the five push buttons and their associated functions.

Table 4.5 Pushbutton and Associated Function Calls

Push button	**Function**
Plot	'plot_callback'
Semilogx	'semilogx_callback'
Semilogy	'semilogy_callback'
Bar	'bar_callback'
Loglog	'loglog_callback'

Figure 4.22 shows a newly created plot using the Figure Plotter. The MATLAB 'plot' has its own tools available, allowing complete customization of plot. These tools include the ability to edit axes properties, line properties, and text properties. This means that a title to the plot, label 'x' and/or 'y' coordinates, and turn grids on and off, can be added. The Axes Property dialog box is illustrated in Figure 4.23.

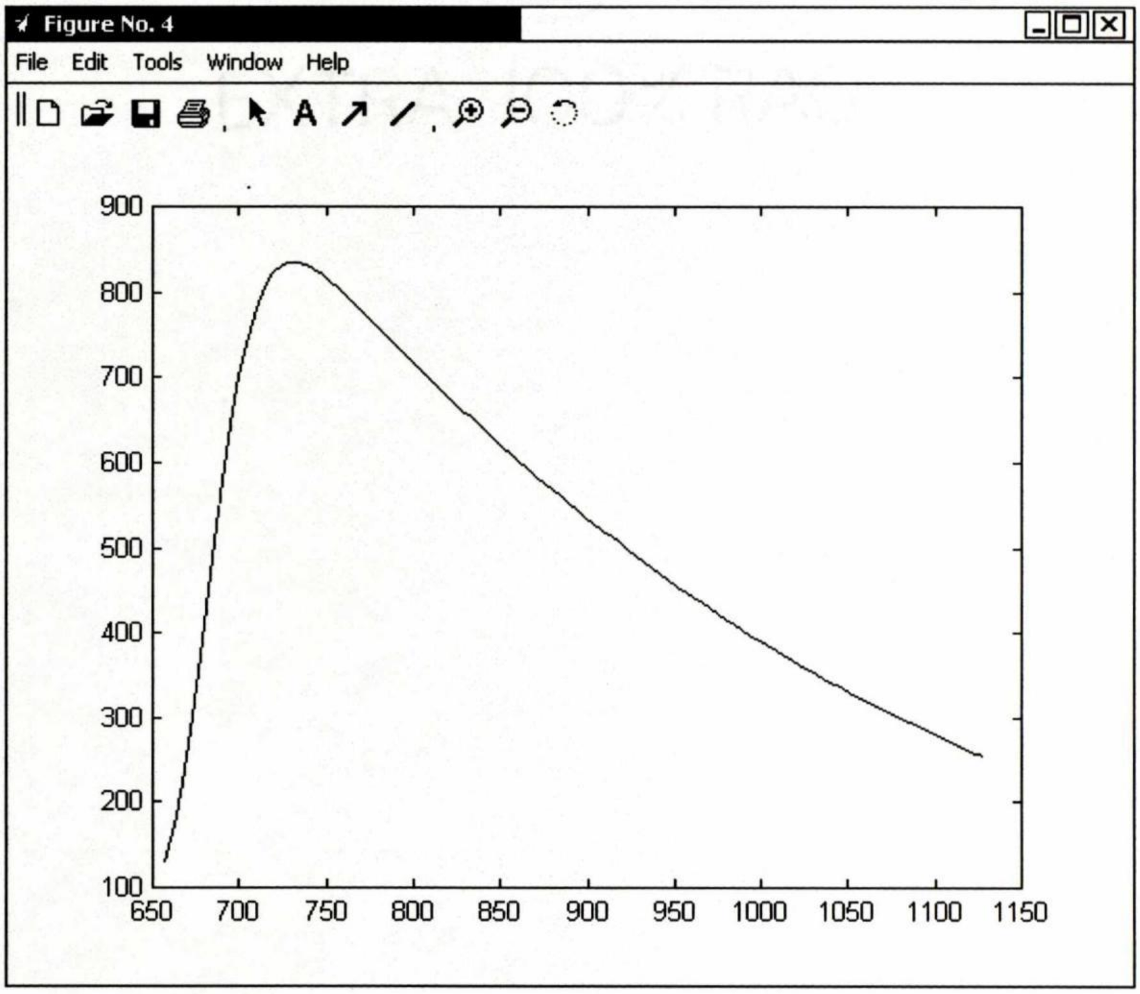
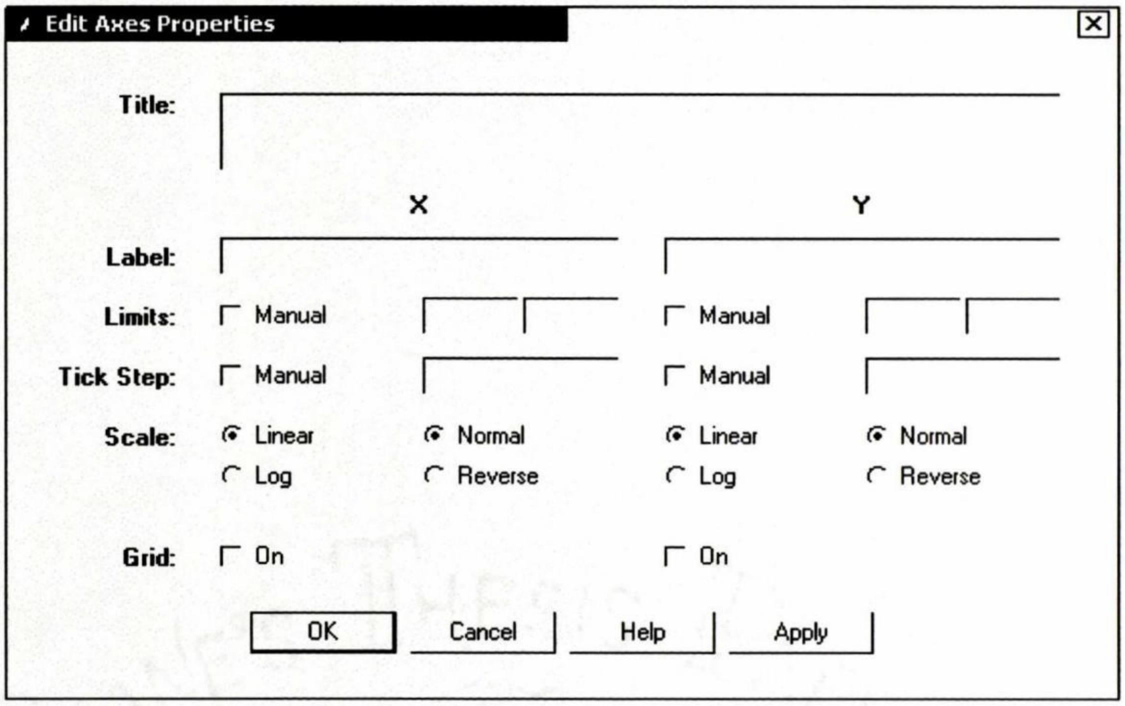


Figure 4.22 Newly Created Plot


Edit Axes Properties

	X		Y	
Title:				
Label:				
Limits:	☐ Manual		☐ Manual	
Tick Step:	☐ Manual		☐ Manual	
Scale:	[✓ Linear] Log	[✓ Normal] Reverse	[✓ Linear] Log	[✓ Normal] Reverse
Grid:	☐ On		☐ On	

Figure 4.23 Edit Axes Properties Dialog box

The only plotting push button that requires only on input is the 'Pie' push button. With pie plotting, the ability to 'explode' a portion of the pie chart exists; the largest slice of the pie chart is pulled away from the rest of the chart.

There are two important facts that the user needs to be aware of. The first has to do with saving the plot. If the user wishes to use the figure list within the Figure Plotter, then the created figure needs to be saved into the same directory as the application. The second has to do with the interaction between the property dialog boxes that open to change properties of the plot and the Windows XP operating system. Windows XP has two types of appearances that can be selected from the 'Display Properties' dialog box. When this selection is set to 'Windows XP style', the user will have a very hard time being able to make changes to any of the dialog boxes when trying to customize a plot. Therefore, it is very highly recommended that if the user plans on creating customized plots, then set the style within the 'Appearance section' of the 'Display Properties' to 'Windows Classic Style'.

MATLAB is a command line oriented application, where users have the capability of entering data into the workspace from a keyboard. With this in mind, the advanced user knows that he/she has the added capability of adding temporary variables to the workspace. But that alone does not allow the user to use that variable within the Figure Plotter. Therefore, the push button 'Update Variable List' is implemented to update any variables that may have been added to the workspace. It also will change both variable names in the 'Variables Selected' section to 'NONE'.

When a new plot is created and saved into the appropriate application directory, the 'Select a Figure' list box is not automatically updated. The 'Update Figure List' push button was created and implemented to searches the currently running application directory to find any files that has the (.fig) extension. All of these filenames are collected and placed into the 'Select a Figure' list box. Table 4.6 lists the functions that are used in conjunction with the Figure Plotter.

Table 4.6 Figure Plotter Interface Functions

Function	Purpose
Update_figbox	Checks and updates the list containing previously created figures.
Update_listbox	Checks and updates the list containing previously created variables.
varDisplay	Controls specific sections of the Figure Plotter Interface. Checks to see if a plot is open.
plot_callback	Retrieves values to be plotted by using the 'get_twovars' function and uses the MATLAB plot() function to perform actual plotting.
get_twovars	Retrieves values to be plotted when using the 'plot_callback' function.
get_onevar	Retrieves values to be plotted when using the 'pie_callback' function.
pie_callback	Retrieves values to be plotted by using the 'get_onevar' function and uses the MATLAB pie() function to perform actual plotting. Also uses the MATLAB explode() function to separate the largest section of the pie chart.
figDisplay	Used to open a plot after it has been selected from the figure list.
detect_figure	Enables the 'Add Variables' and 'Replace Variables' radio buttons on the Figure Plotter.
figDelete	Searches for the figure that is selected within the figure list and deletes the actual figure file using the MATLAB delete() function.

4.6 Chapter Summary

This section described the implementation process that was required to bring GPET into existence. Individual interfaces' were discussed along with the controlling methods associated with that interface. Each interface that exists or is created belongs to or becomes an architectural component within the entire system. Control of the system comes from the various methods (functions) belonging to each component of the overall architecture. In the next section, we will test and evaluate the entire system for reliability as well as limitations.

CHAPTER V

GPET TESTING AND EVALUATION

5.1 Introduction

This section will look at the testing method used to validate the GPET software system. Testing of any software package requires a detailed master test plan, and this software is no different. The primary focus of the master test plan is to evaluate all aspects of GPET.

The chapter begins with a discussion on verification and validation. Specifically, the discussion will include what verification and validation is and the differences between the two types of testing. Verification and validation is covered in §5.2. The discussion will then turn to the various tests plans created for GPET. This all begins with a master test plan.

The master test plan consists of two parts. Part one is a table that contains the areas of functionality, and the second part comprises three phases of testing. The Master Test Plan is discussed in §5.3. Phase one of the Master Test Plan, which is covered in §5.3.1, is based on a prototype application developed from scratch. Phase two is based on an existing GIFTS-related model application that will be modified for the GPET system, which will be discussed in §5.3.2. Phase three will evaluate GPET with the installation of the two applications and will be discussed in §5.3.3. Each individual phase will not evaluate the entire software system. It is the culmination of all three phases that achieves overall validation. The chapter summary highlights the chapter, which is covered in §5.4.

5.2 Verification and Validation

Verification and Validation is a software engineering discipline that helps build quality into software [3]. Verification is accomplished during each life-cycle phase to ensure that requirements of the previous life-cycle phase are being met. Validation involves the conformation of the software at the end of development to ensure that the software does what it is suppose to do [3]. The previous version of the tool cannot be used to validate GPET because of the amount of functionality implemented within GPET that is not supported in the previous version. Validation of GPET is derived from ensuring that the requirements within the Problem Statement are met. Some of the benefits of V&V include uncovering potential errors early, ongoing evaluation against system requirements, gives feedback to management for the purpose of knowing the progress of development, and gives the user a step-by-step view of the software so that early fixes can be accomplished [3].

The scope of this chapter is to test the overall functionality of the GPET software. To that end the Master Test Plan was developed. The purpose of the Master Test Plan is to test the overall functionality of the GPET software by creating a table of areas of functionality along with test phases. Each test phase will validate specific areas of functionality within the software, and the cumulative phases will validate the overall package. An important point to note is that testing is focused on the GPET software, not on the accuracy of the calculations performed within the model applications.

5.3 Master Test Plan

The purpose of creating the Master Test Plan for this project is to define all the areas within GPET to be tested. The Master Test Plan for this GPET is a 30-point test plan, where the three phases will test the 30 points. During the three phases, there will be some overlapping, where some areas of the master plan will be tested multiple times. The multiple testing will only serve to further validate that particular area. Table 5.1 lists the 30-point testing of the GPET software.

Table 5.1 Master Test Plan Areas for GPET Software

1	Application Template	16	Create Result Interface
2	Variable Template	17	Test max number of Result Interfaces
3	Installing application	18	Test Result internal variable names
4	Uninstalling application	19	Edit an exiting Input Interface
5	Selecting an application	20	Edit an existing Result Interface
6	Preselecting an experiment	21	Reinstall an Input Interface
7	Use default values	22	Reinstall a Results Interface
8	Use values from another experiment	23	Run an experiment
9	Use values from another application	24	View Input values using Input Interface
10	Add an experiment	25	Edit values using Input Interface
11	Check for existing experiments	26	View results using Results Interface
12	Selecting an experiment	27	Plot variables with Figure Plotter
13	Create an Input Interface	28	Display an existing plot
14	Test max number of Input Interfaces	29	Edit an existing plot
15	Test Input internal variable names	30	Delete an existing plot

Using Table 5.1 as reference, Table 5.2 illustrates the areas of the Master Test Plan that will be tested within each phase. This table will be reference within its respective section.

Table 5.2 Areas tested within each phase

Phase/ Area	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1 (§5.3)	●	●	●			●	●			●		●	●		●
2 (§5.4)	●	●	●			●	●	●		●	●	●	●		●
3 (§5.5)	●	●	●	●	●	●			●	●	●	●	●	●	●
Phase/ Area	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1 (§5.3)	●		●					●	●	●	●				
2 (§5.4)	●		●					●	●	●	●	●	●	●	●
3 (§5.5)	●	●	●	●	●	●	●	●	●		●				

5.3.1 Phase One: Test of GPET Primitives

As described in earlier chapters, the model application is essentially a computational engine within GPET, meaning that all it does is calculate values from equations. Even though GPET was developed for GIFTS, as long as any model application conforms to the Application Template, it will work within GPET.

In this phase a model application is designed and implemented from scratch, and after evaluating it as a standalone, the application will be implemented within GPET by using the Application Template. Figure 5.1 illustrates the architecture of the application as a standalone. This test phase will test all areas listed in the first row of Table 5.2 Note that not every area listed in this phase will be explicitly mentioned. The areas not discussed were implicitly tested while testing the areas that were considered for discussion.

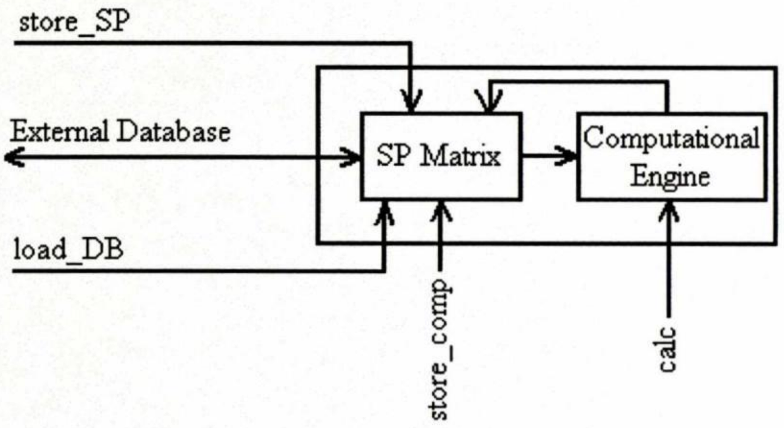


Figure 5.1 Protoytype Application Architecture (Standalone Form)

Once the application has been developed as a standalone, it will be run several times for the purpose of collecting data. These runs will correlate to different experiments within the shell, and will be collected for reference. Once the standalone application is run a number of times, it will be installed into GPET using the Application Template. Several experiments are created within the application that will parallel the number of runs performed as a standalone. The application will be run a number of times to produce various results. These inputs and results will be compared to the standalone application input and results.

The standalone application begins by defining the directory and filename of the location of the database. The database file is opened, and the data is loaded into the workspace under the variable called 'SP'. 'SP' is a 20x20 matrix. For this application, focus will be placed within a small portion of that matrix. On the input side the application consists of 20 values labeled 'a' through 't'. The input variables are listed as constants within the application. So, in order to run the application with different values, the application itself must be modified. For this phase it was decided that five tests would be sufficient to test the areas listed earlier. Table 5.3 lists the tests and the associated input values used.

Table 5.3 Test Numbers and Associated Initial Values

Test	Values
U1	0.5 – 10.0 in increments of 0.5
U2	-0.5 – -10.0 in increments of 0.5
U3	0.25 – 5.0 in increments of 0.25
U4	-0.25 – -5.0 in increments of 0.25
U5	-2.5 – 2.5 in increments of 0.25 (no zero value)

After input values are assigned they are then stored into SP(1,1) through SP(1,10) and SP(2,1) through SP(2,10). The calculations on the variables are now performed in four sections. Table 5.4 lists the manner in which the calculations are taking place.

Table 5.4 Output Variable Names and Associated Operation

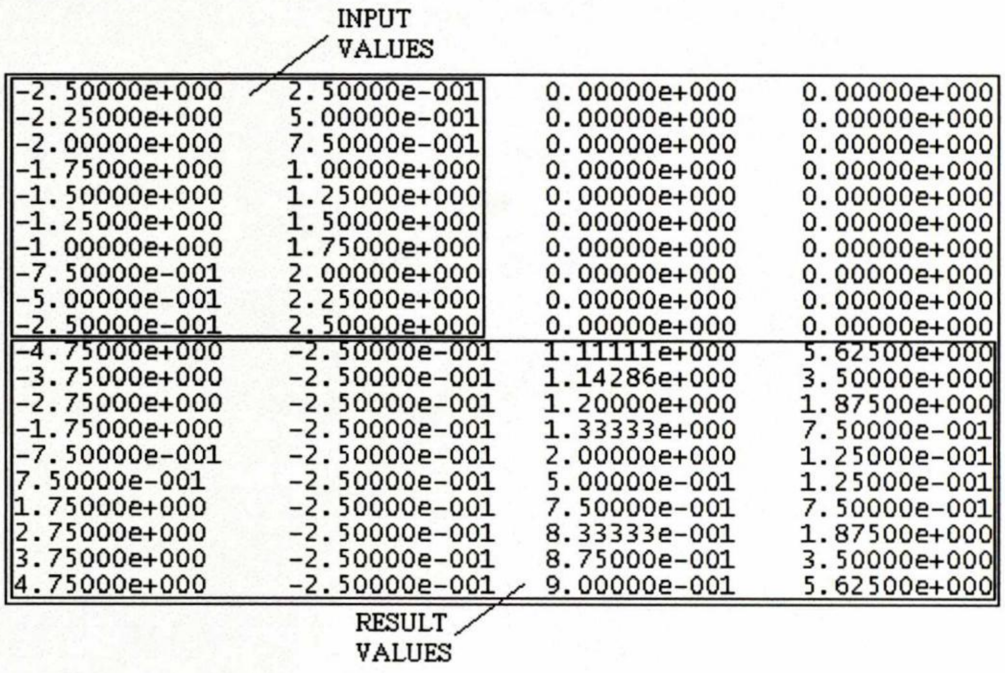
Variable Names	Operation
aa through jj	Add
aaa through jjj	Subtract
aaaa through jjjj	Divide
aaaaa through jjjjj	Multiply

Once the calculations are completed, results are assigned to specific elements of the 'SP' matrix. Table 5.5 lists the variable names and the areas of the 'SP' matrix that the variables are stored in.

Table 5.5 Output Variable Names and Associated SP Locations

Variable Names	'SP' locations
aa through jj	SP(1,11) through SP(1,20)
aaa through jjj	SP(2,11) through SP(2,20)
aaaa through jjjj	SP(3,11) through SP(3,20)
aaaaa through jjjjj	SP(4,11) through SP(4,20)

Finally, the database is opened once again. This time the values that were stored into the 'SP' matrix are now saved into the database file, and then the file is closed. Figure 5.2 illustrates Test U5 inputs and results stored within a database text file.


INPUT VALUES
RESULT VALUES

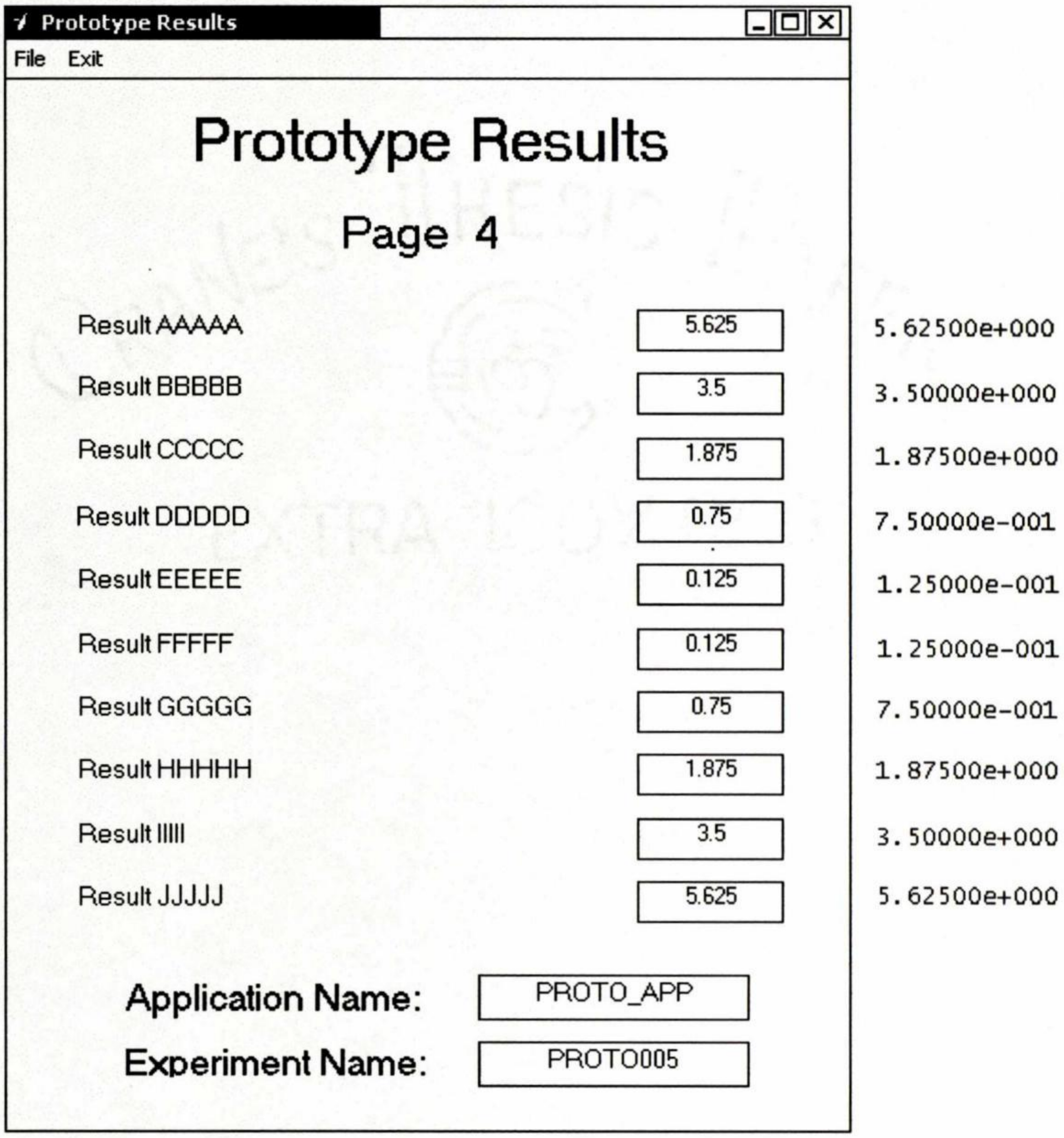
```
-2.50000e+000    2.50000e-001    0.00000e+000    0.00000e+000
-2.25000e+000    5.00000e-001    0.00000e+000    0.00000e+000
-2.00000e+000    7.50000e-001    0.00000e+000    0.00000e+000
-1.75000e+000    1.00000e+000    0.00000e+000    0.00000e+000
-1.50000e+000    1.25000e+000    0.00000e+000    0.00000e+000
-1.25000e+000    1.50000e+000    0.00000e+000    0.00000e+000
-1.00000e+000    1.75000e+000    0.00000e+000    0.00000e+000
-7.50000e-001    2.00000e+000    0.00000e+000    0.00000e+000
-5.00000e-001    2.25000e+000    0.00000e+000    0.00000e+000
-2.50000e-001    2.50000e+000    0.00000e+000    0.00000e+000
-4.75000e+000   -2.50000e-001    1.11111e+000    5.62500e+000
-3.75000e+000   -2.50000e-001    1.14286e+000    3.50000e+000
-2.75000e+000   -2.50000e-001    1.20000e+000    1.87500e+000
-1.75000e+000   -2.50000e-001    1.33333e+000    7.50000e-001
-7.50000e-001   -2.50000e-001    2.00000e+000    1.25000e-001
7.50000e-001    -2.50000e-001    5.00000e-001    1.25000e-001
1.75000e+000    -2.50000e-001    7.50000e-001    7.50000e-001
2.75000e+000    -2.50000e-001    8.33333e-001    1.87500e+000
3.75000e+000    -2.50000e-001    8.75000e-001    3.50000e+000
4.75000e+000    -2.50000e-001    9.00000e-001    5.62500e+000
```

Figure 5.2 TEST U5

Once all the data is collected from the standalone model application, steps are taken to incorporate the model application into the GPET system. This includes the use of the application template to format the application into, as well as creating the variable initialization file for use within the Input/Result Interface Creator. Once these files are created and placed into its respective application directory, the model application was installed the application into the GPET shell. From that point, two input interfaces were created that will allow the modifying of experiment inputs prior to running the application. Four result interfaces were created to provide the ability to view the outcome of selected experiments and compare those results with the standalone model application. Next, five experiment/result object pairs were created using the 'Default Values' selection from the Experiment Selection Interface. This selection allows the use of an external database to provide preset values into each experiment object. Once all five experiment/result object pairs were created, the Input Interfaces were used (two interfaces) to set the input values according to the value set forth in Table 5.3. Once all the values for each experiment are set correctly, each experiment is selected individually and the application is run. Upon completion of the experiment, each of the four Result Interfaces is open separately and the results within the interface is viewed and compared to the database file associated with the standalone application. In all five experiments, the results of the installed application were identical to those of the standalone application. Figure 5.3 illustrates one of the result interfaces displaying the results from running experiment 'proto05'. Next to each result text field is the data that came from running 'TEST U5' of the uninstalled version. Figure 5.2 shows the input and result from running 'TEST U5'.

5.3.2 Phase Two: Test of Pre-Existing Model

In this phase we will use a pre-existing model application called 'SNR_model'. The 'SNR_model' model application is a standalone application, and was actually the foundation model application used in testing the shell in the undergraduate project. Unlike the prototype model application, the 'SNR_model' model application has many dialog boxes that are used to enter data and to review the results of assigned input data, textually, built into the model application. In addition, various MATLAB plot functions have been incorporated into the model application for viewing results in a graphical format.



Prototype Results

Page 4

Result AAAAA	5.625	5.62500e+000
Result BBBBB	3.5	3.50000e+000
Result CCCCC	1.875	1.87500e+000
Result DDDDD	0.75	7.50000e-001
Result EEEEE	0.125	1.25000e-001
Result FFFFF	0.125	1.25000e-001
Result GGGGG	0.75	7.50000e-001
Result HHHHH	1.875	1.87500e+000
Result IIIII	3.5	3.50000e+000
Result JJJJJ	5.625	5.62500e+000

Application Name: PROTO_APP

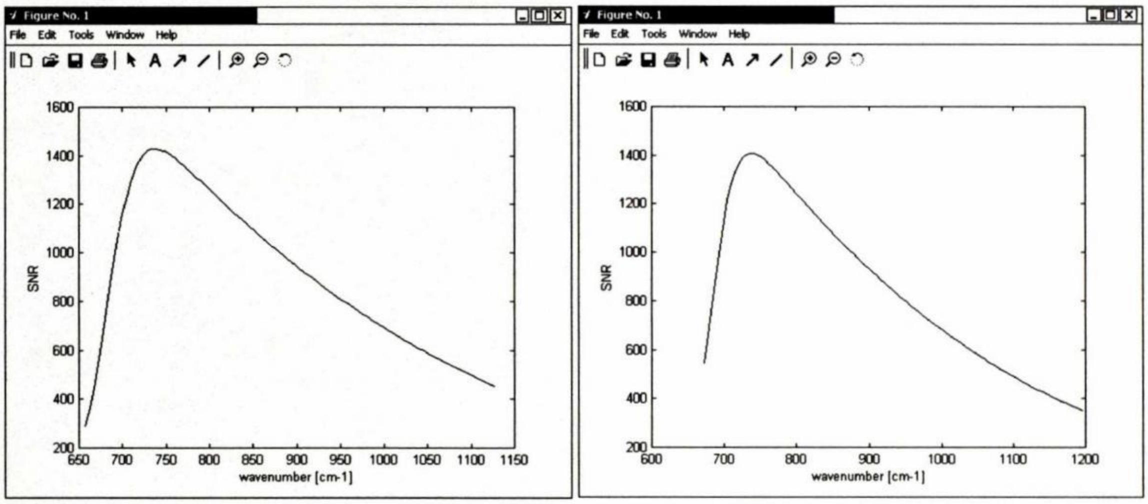
Experiment Name: PROTO005

Figure 5.3 Prototype Results

The standalone version begins by requesting the user for the directory and file name of the external database to retrieve and send data. Five dialog boxes will then appear showing the user what the input value are set at and gives the user an opportunity to change the data. Some calculations are than performed followed by seven different plots (one of which is a pie chart plot). Two more dialog boxes appear after the calculations that show the results of some of the calculations that have taken place. After that two more plots are shown, and finally three more dialog boxes showing results.

Since we have an existing model application, we need to collect data on this standalone version. Specifically, the application will run with values that have been previously assigned by the end user. Then, we will change two input values (W1 and W2), which were chosen at random, and run the application a second time. We will have the ability to change these values through the use of a dialog box that has been hard-coded into the model application. The purpose of this test is not to ensure that the application is working properly, but to provide reference plots to use once the model application is installed into GPET.

Figure 5.4 shows one of the plots when W1 = 650 and W2 = 1130 (Figure 5.4.a) and W1 = 700 and W2 = 1200 (Figure 5.4.b). This will be used after the model application is installed. We will once again plot SNR vs. wavenumber with the two sets of values, do a screen capture, change the two sets of values, and plot SNR vs. wavenumber.

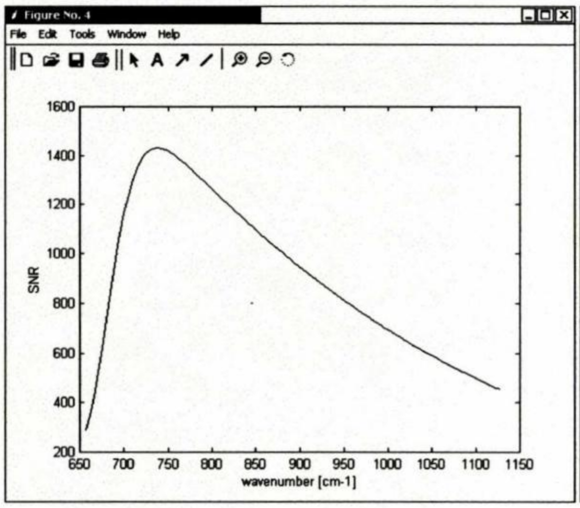


(a) W1 = 650, W2 = 1130 (b) W1 = 700, W2 = 1200

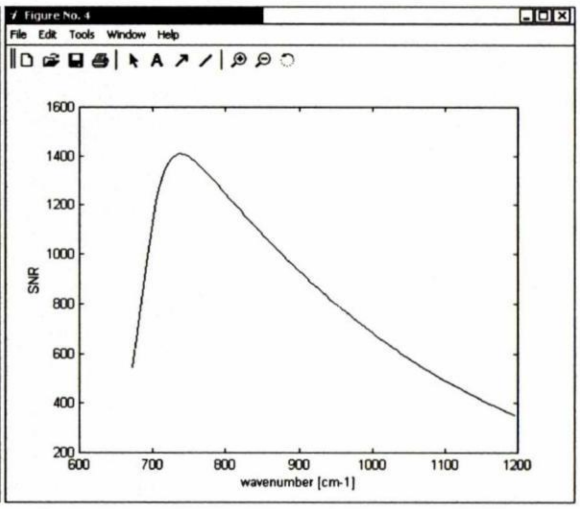
Figure 5.4 SNR Plots

After collecting data in the standalone version, we will then install the model application into the GPET system. This test phase will test all areas listed in the second row of Table 5.2. As with §5.3, not every area listed will be discussed explicitly in this section, but at some point each area was tested. The standalone version of the model application uses a database format that is not consistent with the database format of the GPET system. Therefore, the database needs to be formatted so that it fits the 20x20 format that the system uses. Then, the application can be started and can create the two experiments that are needed. The first experiment will contain all the original values, and the second will contain the revised W1 and W2 values.

When the appropriate values have been placed into the two objects, the first experiment is selected to run (with the original values). After running the experiment, the Figure Plotter is used to create the plot similar to that in the standalone version. The variables that will be selected will be F1 first followed by SNR. These are the two variables that the user had plotted in the standalone application. When selecting the PLOT function, a figure will appear with the appropriate plot that can be manually customized to have the same appearance as the plot produced by the standalone. The second experiment is selected and the same procedures are followed as was for the first one. Figure 5.5 illustrates the results of the two experiments with experiment 'revsnr001' shown in Figure 5.5.a and 'revsnr002' shown in Figure 5.5.b. These two experiments are identical to those in Figures 5.4.a and 5.4.b respectively. The internal variables used to create the plots were also compared directly and were found to be identical as well. A point to keep in mind is that the real test within these three phases is to ensure that the shell functions properly. In all three phases, the model applications are not being tested.



(a) 'revsnr001'



(b) 'revsnr002'

Figure 5.5 Revised SNR Plots

5.3.3 Phase Three: Compositional Model Applications

One of the features built into the GPET software is the ability to take the results from one application and feed them to the inputs of a different application. This functionality may be used in the event that the end user is designing a modular-based application. The application would be divided into smaller sub-applications. The user could then easily collect data for each sub-application as well as apply the results to follow on sub-applications. This phase focuses on testing this particular functionality. This test phase will test all areas listed in the third row of Table 5.2, and as with the previous test phases, not every area will be discussed, however, they were tested at some point during the overall phase testing. Figure 5.6 illustrates the interfacing of two applications.

A very interesting question would be: "If the results of one application contain 40 values, but the input to the following application only takes 20 inputs, then how are 20

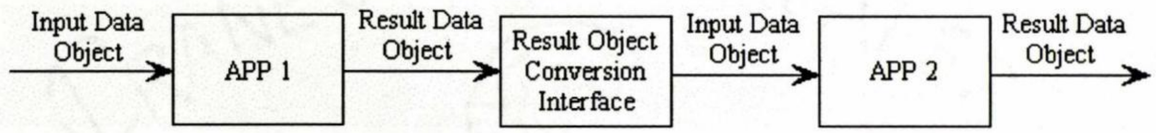


Figure 5.6 Two-Application Interfacing

input variables selected from the possible 40 variables?" In order to answer this question the Application Template for the application receiving the input from a previous result needs to be examined. In the Application Template, there is a section that assigns experiment object values to the application variables. An example of this assignment would be

w1 = get_value (expinput, 'v01');

where 'w1' is an application variable, 'expinput' is the experiment object, and 'v01' is a field value within the object that gets assigned to 'w1'. Using the 'get_value' function, any of the 200 variables ('v01' through 'v200') within an experiment object can be retrieved and assigned to an application variable. It does need to be understood that the experiment object that is being referred to contains the 40 values (in this test) from a previously created result object. However, any of the 20 values can be chosen from the current application by the way the current application is developed. In other words, only 20 values are needed for the input to the current application from a possible 40 values.

In this test two applications are required. The prototype application will be used along with a modified version of the prototype. Note that there will be no running of the standalone modified prototype application. Recall in §5.3 the prototype application has with 20 input variables and 40 output variables, and that it performed some basic arithmetic and then stored the results with a result object. The modified version of the prototype application also takes 20 input variables and produces 40 output values. The only difference between the two applications is the order of the calculations. The modified version performs subtraction, multiplication, addition, and subtraction in that order.

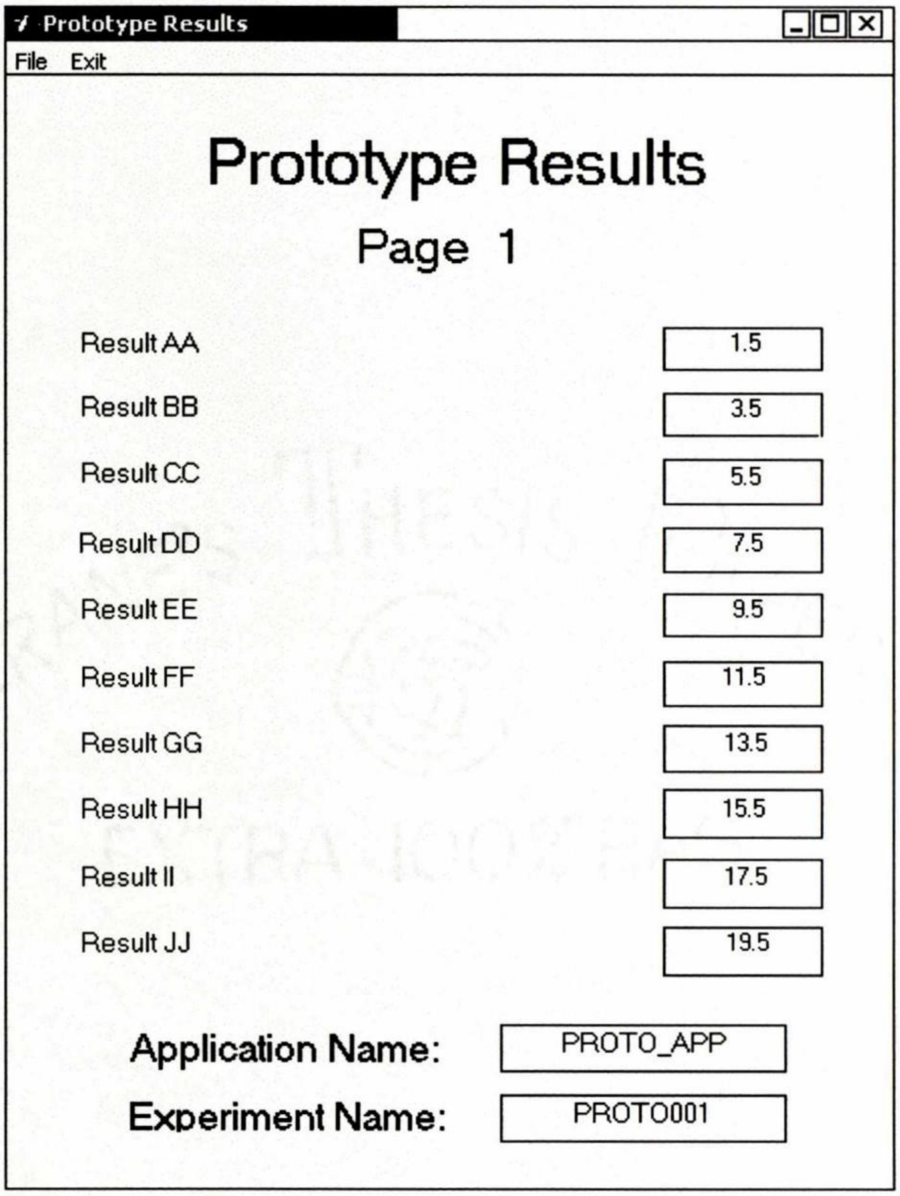
Testing starts by first creating a particular set of 20 input values for the original prototype application. We will focus only on the portion of the application that performs addition. This will correlate to pages one and two of the Input Interface, and page one of the Result Interfaces of the prototype application. Table 5.6 shows the particular values assigned, the variables used in the addition, and the results of the operation.

Table 5.6 Input Variable Assignments

Input Variables	Values assigned	Operation	Result
a, b	0.5, 1.0	a + b	1.5
c, d	1.5, 2.0	c + d	3.5
e, f	2.5, 3.0	e + f	5.5
g, h	3.5, 4.0	g + h	7.5
i, j	4.5, 5.0	i + j	9.5
k, l	5.5, 6.0	k + l	11.5
m, n	6.5, 7.0	m + n	13.5
o, p	7.5, 8.0	o + p	15.5
q, r	8.5, 9.0	q + r	17.5
s, t	9.5, 10.0	s + t	19.5

Once these values have been placed into the experiment for the primitive application, we then run the experiment. Figure 5.7 illustrates page one of the results for the prototype application. The results within this interface are the same as those within Table 5.6.

The next step is to take 20 calculated values from the 40-value result object of the original prototype application and use those values as input to a new experiment within the modified prototype application. The Application Template for the modified model application was designed so that the first 20 result values from the result object will be applied as input. Table 5.7 shows the inputs being used, the operation to be performed on them, and results of the operation.



Prototype Results

Page 1

Result AA	1.5
Result BB	3.5
Result CC	5.5
Result DD	7.5
Result EE	9.5
Result FF	11.5
Result GG	13.5
Result HH	15.5
Result II	17.5
Result JJ	19.5

Application Name: PROTO_APP

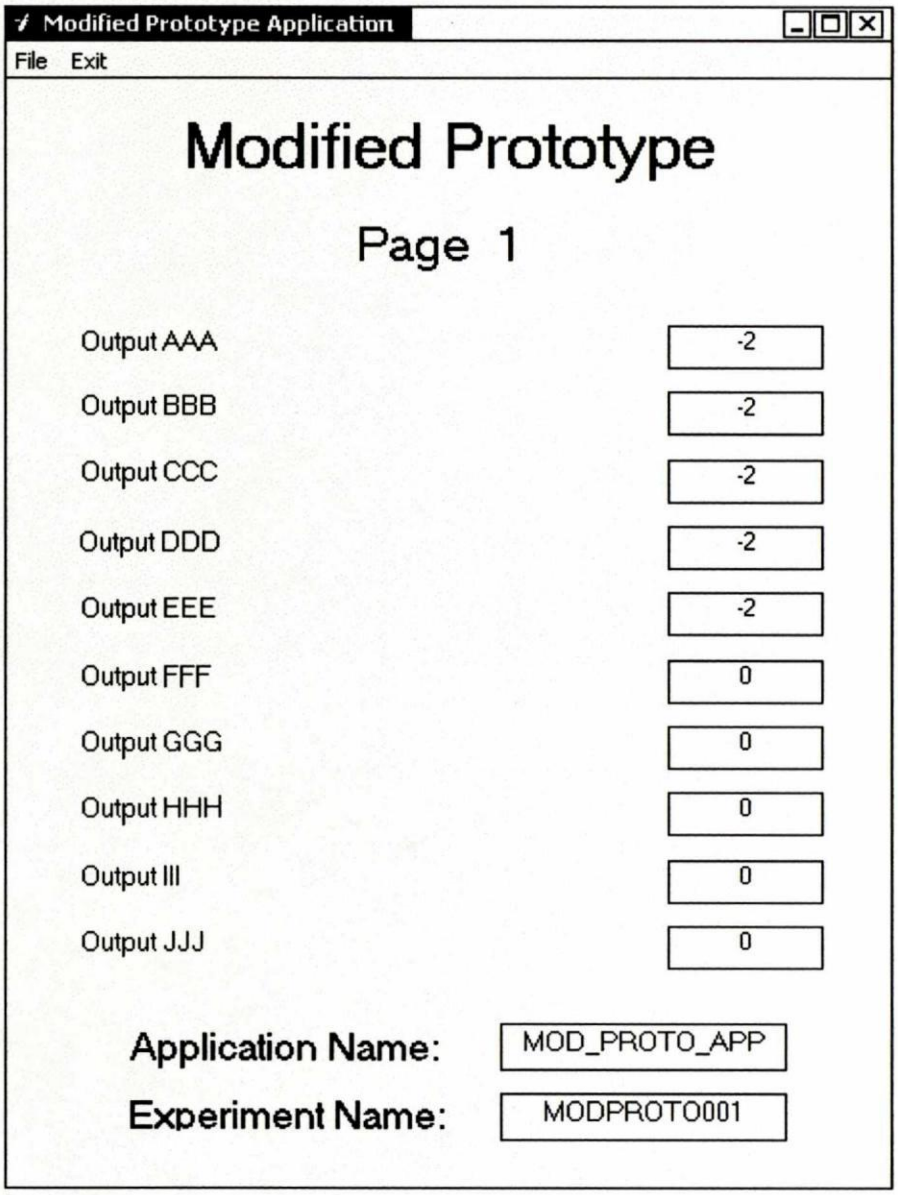
Experiment Name: PROTO001

Figure 5.7 Prototype Results Page 1

Table 5.7 Subtraction Results from Modified Model Application

Input Variables	**Values assigned**	**Operation**	**Result**
a, b	1.5, 3.5	a – b	-2.0
c, d	5.5, 7.5	c – d	-2.0
e, f	9.5, 11.5	e – f	-2.0
g, h	13.5, 15.5	g – h	-2.0
i, j	17.5, 19.5	i – j	-2.0
k, l	-0.5, -0.5	k – l	0
m, n	-0.5, -0.5	m – n	0
o, p	-0.5, -0.5	o – p	0
q, r	-0.5, -0.5	q – r	0
s, t	-0.5, -0.5	s – t	0

The subtraction part of the modified model application will be considered. Figure 5.8 shows the results of the 10 subtraction operations, which are identical to those in Table 5.6.



Modified Prototype

Page 1

Output AAA	-2
Output BBB	-2
Output CCC	-2
Output DDD	-2
Output EEE	-2
Output FFF	0
Output GGG	0
Output HHH	0
Output III	0
Output JJJ	0

Application Name: MOD_PROTO_APP

Experiment Name: MODPROTO001

Figure 5.8 Modified Prototype Results Page 1

5.4 Chapter Summary

In this chapter, the testing and evaluation of the GPET software was discussed. The chapter begins with a discussion of Verification and Validation, where the difference between the two terms is brought forth. Next, a master test plan was developed that lists 30 areas of functionality. The master test plan is broken down into three phases, where each phase tests a portion of the software, and the culmination of the three phases tests all areas of the Master Test Plan.

Phase One of the Master Test Plan uses a prototype model application that performs simple arithmetic operations. The model application was created as a standalone application, and was run a number of times for the purpose of collecting data for reference. The prototype model application was then installed into GPET, and was run a numbers from within the software. The results were collected from the shell and compared to the standalone version, and the results were found to be identical.

Phase Two of the Master Test Plan used an established model application designed specifically for GIFTS. It was delivered as a standalone model application. The standalone version was run twice, once with original database values, and a second time with two variables altered. The original model application contained a number of MATLAB plots, where one of the plots was selected for comparison between the standalone version and the installed version. The selected plot from the standalone and installed versions was captured twice and used for comparison, where the comparison of the plots revealed identical results.

The purpose of Phase Three was to demonstrate the ability to take the results of one application and use them as input into a completely different application. In the testing we used the prototype application along with a modified version of itself. The two applications take 20 inputs and produce 40 results. In the prototype application, focus was placed on the portion of the application that performed the addition operation. Manual calculations were performed on the values used in the addition operation, and those results were compared with the results collected from the installed prototype application, and those results were found to be identical. These results were then applied as input to the modified prototype application, where only the subtraction part of that application was considered. Again, manual calculations were performed and collected, followed by running the modified model application and collecting those results. The comparison of the manual calculations against the results of running the installed application yielded identical results.

CHAPTER VI

CONCLUSION, LIMITATIONS, AND FUTURE RESEARCH

6.1 Conclusion

This thesis concentrated on the research, design, development, and implementation of an object-oriented approach in creating a Generic Performance Evaluation Tool (GPET) in software using the MATLAB programming language. The research for the project was funded through NASA-Langley Research Center. The goal of the project was to provide an object-oriented environment in which to perform experimental analysis and to review experimental results in a textual and graphical format. The areas involved included conceptual modeling, Software Engineering, Graphic User Interface Design concepts, Test Validation, and MATLAB programming.

The foundation to modeling is the conceptual model, which serves as a knowledge base to build upon [9]. Three conceptual models were developed, which were the Systems Model, Operating System Model, and the Shell Model, to give different perspectives on the direction in which software development could go. From the three models the Shell model provided the best development path, mainly due to the existence of the software that was implemented as an undergraduate project that could be built upon.

The next stage was to use the Top-Down development process, which involves Problem Analysis, Program Design, and Program Implementation. The problem statement provides high-level information on what the software is suppose to do. Program Design requires taking the problem statement, and produce smaller, more specific subproblems, until a point is reached the subproblems cannot be broken down any further. At this point algorithms can be produced. The final step, Program Implementation, takes the algorithms and implements them in a specific programming language. The Top-Down design process is not strictly a single iterative process. At any point within the design process the problem statement may change, and those changes need to follow through the process.

Graphic User Interface design played an important role behind the scenes in the development of GPET. One of the largest elements in successful software is to ensure that the end user is involved in the design process. With GPET, the developer and user worked closely to ensure that issues brought forth by the user were being addressed and resolved by the developer. Design issues that could be important to the user are color, layout of controls, understandability of controls, and redundancy. It takes more than good code writing to make a great software package, and not receiving feedback from the audience using the software will more than likely lead to failure.

Another important role in the software development process is the testing of the software. Testing involves the verification and validation of the software. As was mentioned, verification occurs during each phase of software development, where testing in one phase verifies the previous stage. On the other hand, validation is performed on the entire package. In this thesis, focus was placed on the validation of the entire package. The validation of GPET required a 30-point master test plan. The test plan was divided into three test phases, where each phase tested certain points of the master test plan with overlapping of some of the points. All three phases combined tested all 30 areas of functionality.

The programming of GPET involved learning how to program using MATLAB, which included object-oriented programming. Comparisons were made between object-oriented programming in MATLAB and C++, where some similarities and many differences between the two languages were discussed. Topics of discussion included classes, objects, and inheritance.

In concluding, the problem to solve was no contained environment to evaluate GIFTS instrument analytical model applications in MATLAB, and to solve the problem required Modeling, Top-Down Design Process, GUI Design, along with Verification and Validation to move from concept through implementation.

6.2 Software Problems, Limitations and Constraints

The research, design, development, and implementation of GPET was a great success. However, there are problems and limitations in the software. One of the problems involves the Windows XP operating system. Specifically, the problem becomes apparent when using the 'plot editing' function when using the Figure Plotter. When the operating system is set to display windows using the 'Windows XP Style', the user will have a problem in setting some of the values within the 'plot editing' dialog window. In order to resolve this limitation the 'Windows and Buttons' selection within the Display Properties dialog box for Windows XP needed to be set to 'Windows Classic Style'. It is currently unknown what the source of the problem is; however, there are two possible sources. First, the specific version of MATLAB used (Release 11, version 5.3 was used to implement GPET) may have compatibility problems with XP. Second, the operating system itself could be the culprit.

There are also software problems that exist within GPET. One of the problems occurs when an experiment is added to or deleted from an application. In order for an experiment to be added or deleted, code had to be built into GPET to cause it to completely exit. A possible explanation for this is that the only way for the Application Interface to be updated is to first close its interface window, and to update system variables requires closing the main window. GPET must then be restarted.

Another problem occurs when installing/uninstalling applications to/from GPET. In order for an application to be installed, a unique directory must exist with the application being installed. In order to add the directory to the MATLAB path, the path browser must be started. This is an automatic operation when an installation occurs, however, the user is still forced to save the path and close the path browser window manually. Numerous hours were spent on trying to eradicate the manual steps; however, the manual steps must still be performed. The same situation occurs when uninstalling applications as well. An associated intermittent problem that occurs is that sometimes the path browser does not display for updating. If the path browser does not display, then the user is forced to manually open the path browser and enter the directory of the application being installed.

Like any software tool, GPET has a number of constraints. One constraint is the number of input and result variables contained within each object. GPET is constrained to 200 input and result variables each. Another constraint is the external database. It must be set up as a 20x20 matrix. The user is also constrained to 10 input and 10 result variables per Input and Result Interface when creating such user-defined interfaces, and that will constrain the user to a maximum of 20 input and result interfaces each.

6.3 Future Work

In order for GPET to function it must be used within the MATLAB environment. An improvement to the current software would be to implement GPET into a standalone executable. The current version of MATLAB does have specific libraries that allow for development into a standalone software package; however, these libraries are available as an add-on module that can be purchased from MathWorks.

Even though producing a standalone software package is in the realm of possibility, further research is needed on how to develop the model applications so that they can be utilized by the standalone version.

REFERENCES

[1]. Frank L. Friedman, Elliot B. Koffman, Problem Solving, Abstraction, and Design Using C++, Addison-Wesley Publishing Company, 1994.

[2]. "Principles of good GUI Design" http://axp16.iie.org.mx/Monitor/v01m03/ar_ihc2.htm.

[3]. Dolores R. Wallace and Roger U. Fujii, Software Verification and Validation: An Overview, IEEE Software pp 10-17, May 1989.

[4]. James Mengert and Denise Linthicum, User Interface and Signal Processing for Modeling an Imaging Fourier Transform Spectrometer, ECE486 Senior Design Project Report, December 2000.

[5]. "The Mathworks HelpDesk" http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/ matlab_prog.shtml.

[6]. Frank M. Carrano, Data Abstraction and Problem Solving with C++, Walls and Mirrors, Addison-Wesley Publishing Company, 1995.

[7]. "NASA to fly high-tech Earth observer" http://www.enn.com/enn-news-archive/2000/01/01082000/gifts_8841.asp.

[8]. Gerard J. Holzmann, From Code to Models, Proceeding of the Second International Conference on Application of Concurrency to System Design (ACSD'01) pp 3-10.

[9]. Paul A. Fishwick, Simulation Model Design and Execution, Prentice-Hall, 1995.

[10]. Reinhard Beer, Remote Sensing by Fourier Transform Spectrometry, John Wiley & Sons, Inc, 1992.

Supplemental Sources Consulted

[1]. Ralph Highnam, Michael Brady, Model-Based Image Enhancement of Far Infrared Images, IEEE Transactions on Pattern Analysis and Machine Intelligence pp 410-415, April 1997.

[2]. Dan R. Olsen, Jr., Developing User Interfaces, Morgan Kaufmann, 1998.

[3]. Joel Spolsky, User Interface Design for Programmers, Springer-Verlag, 2001.

[4]. Jeongwon Baeg and Yoshiaki Fukazawa, A Dialog-oriented User Interface Generation Mechanism, pp 310-317, Proceedings of the 3rd Asia-Pacific Software Engineering Conference (APSEC '96) pp 310-317.

[5]. Laura A. Valaer and Robert G. Babb II, Choosing a User Interface Development Tool, IEEE Software pp 29-39, July/August 1997.

[6]. Sumner P. Davis, Mark C. Abrams, James W. Brault, Fourier Transform Spectrometry, Academic Press, 2001.

[7]. "Michelson Interferometer" http://www.physics.berkeley.edu/courses/labs/7c/7cmicls01/7cmicls01.pdf.

APPENDIX

APPENDIX A – MATLAB SOURCE FILES

The source files for the GPET software are too large to incorporate within this thesis. All source files will accompany the thesis on a separate CD, and all files will be in the MATLAB .m format.

APPENDIX B – GPET OPERATING MANUAL

B.1 Background

The concept of GPET itself was derived from an undergraduate project that focused in part on creating an environment (shell) in which model applications relating to the Geosynchronous Imaging Fourier Transform Spectrometer (GIFTS) project could run using MATLAB. The GIFTS project is a NASA-Langley Research Center sponsored project. It is an advanced satellite that will incorporate, among other uses, the ability to make more accurate weather predictions. A report titled "User Interface and Signal Processing for Modeling an Imaging Fourier Transform Spectrometer" goes into detail on how the user interface (GUI) was designed, developed, and implemented.

GPET-GIFTS is a much more advanced version of the software then its predecessor. Many of the newer features will be discussed in detail in the following sections.

B.2 Features

Some of the features of this software package include the ability to have many experiments per application, and to have multiple applications. Also, the user is given the ability to create their own personally designed input and results interface windows. The user is also given the flexibility to producing individualized graphs.

B.2.1 Object-Oriented Experiments

One of the abilities of MATLAB is to create user-defined classes. MATLAB defines a particular manner in which these classes are utilized; however, they are analogous to classes in C++.

Another capability of MATLAB is the use of inheritance. In this software system we make use of inheritance.

In making use of classes and inheritance, we decided to develop the concept of an experiment and a result. An experiment (for our purposes) is the collection of data for processing within an application. A result is the collection of data produced from the execution of an application.

In our system, an experiment and result are two separate objects. Each object has the ability to contain 200 scalar variables. Therefore, in totality, each experiment has the ability of making use of 400 individual variables.

B.2.2 Multiple Applications

GPET-GIFTS is designed to have a multitude of model applications. Each application in turn will have the ability to contain many experiments. As will be explained in more detail in section three, experiments from one application may be used to create experiments within another application.

B.2.3 Input/Results Interface Creator

Having experiment and result objects is all good and well, however, the user would be extremely limited if there was no way of being able to edit inputs prior to running experiments, and being able to view the results of a currently ran application. Therefore, this software tool provides the user the ability to design Input and Result Interface windows.

B.2.4 Figure Creator

In addition to having the ability to view results of an experiment, the user will most likely desire a manner in which to view certain results graphically. Therefore, the ability for the user to create and annotate graphs, were incorporated into the software.

B.3 Details of Operation

In this section we will give a detailed explanation of how to use GPET-GIFTS. The major areas covered include the System Interface, Application Interface, Input/Results Interface Creator, and the Figure Creator.

B.3.1 System Interface

In order to display the System Interface window, MATLAB needs to be running. Once MATLAB has been started, the user must change directory into the system directory, <drive letter>/Research/@expin. Once the user is in this directory, open the System window by typing "GIFTS_MAIN". The System window is displayed in Figure 3.1.

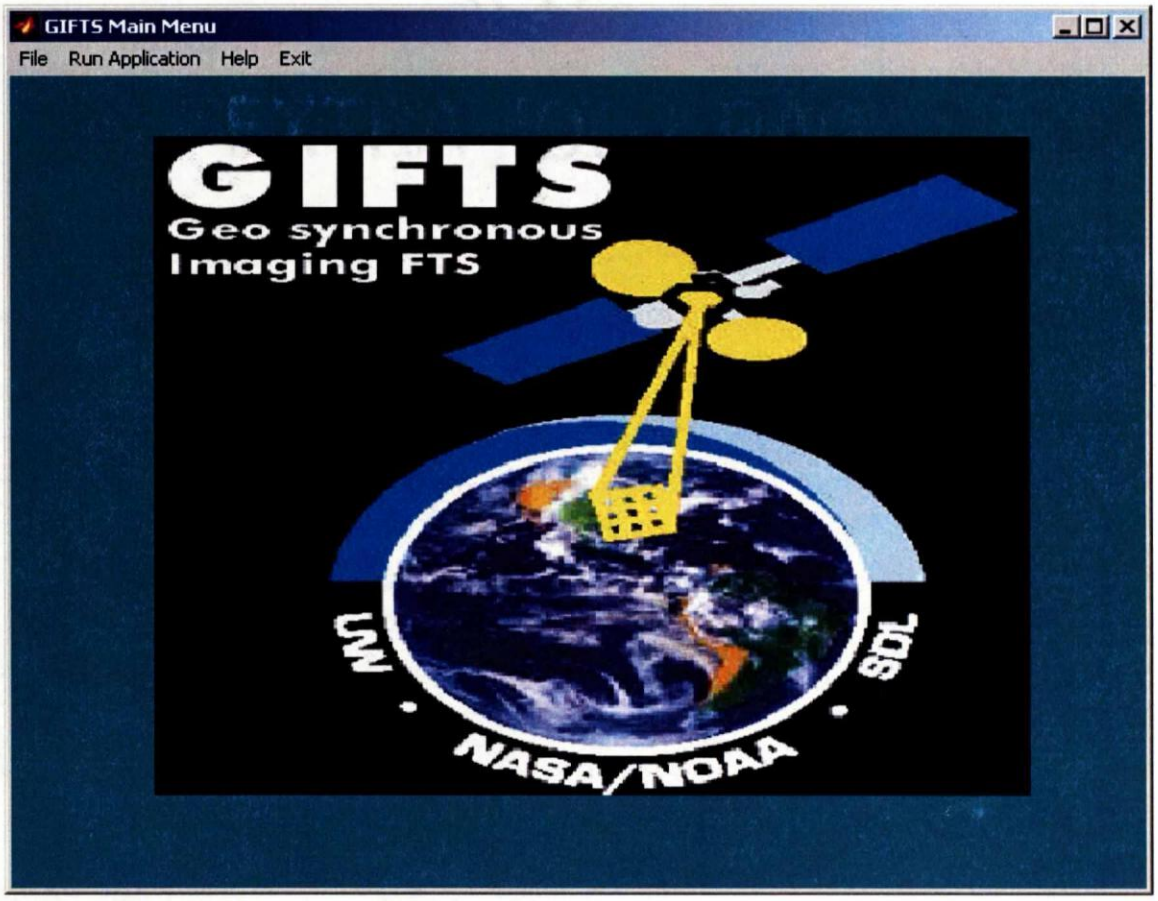


Figure 3.1 GPET-GIFTS Main Menu Interface

B.3.1.1 Application Interface

In order to install an application, the user needs to create a directory for the application, and two files are needed. The application directory is created within the subdirectory '<drive letter>/Research/GIFTS/Application/<application name>/'.

Once the subdirectory is created, they need to be populated with the Application Template, and the Variable Template.

B.3.1.1.1 Application Template

Within the subdirectory, '<drive letter>/Research/blank/' is a file called 'APP_TEMPLATE.m'. This is a template file used to help the user create an application. There are a number of notes within the file that instruct the user on how to use the template. Once these instructions are followed, save the file within the application subdirectory previously created. Give the application file a unique name that clearly identifies the model application. Be careful not to change 'APP_TEMPLATE.m'.

B.3.1.1.2 Variable Template

Within the same directory as 'APP_TEMPLATE.m' is a file called 'variable_init.m'. Within this file are directions on how to modify this file for the user's application. As with the Application Template, save this file to the application subdirectory previously created. The name of the variable template file needs to follow the following format: '(application_name)_init.m'. As an example, if your application file is called 'SNR_MODEL.m', the application initialization file needs to be called 'SNR_MODEL_INIT.m'. Also be careful not to modify 'variable_init.m'.

B.3.1.1.3 Installation

Once the subdirectory has been created for the application, and the associated files have been added to the subdirectory, it is time to install the application using the System Interface. From the 'File' pull-down menu, select 'Install Application'. At this point a dialog box will appear. Enter the application subdirectory and application name within the two sections of the dialog box, than click on 'OK'. At this point the Path Browser interface will open. Select 'File' and than 'Save Path'. Than select 'File' again and select 'Exit Path Browser' to close the Path Browser. At this point the application has been installed.

B.3.1.2 Application Uninstallation

To uninstall an application, select 'Uninstall Application' from the 'File' pull-down menu. Once again a dialog box will appear. Enter the directory and filename in the associated sections of the dialog box, than click on 'OK'. As with the install process the Path Browser interface will open. Select 'Save Path' from the 'File' pull-down menu. Then select 'Exit Path Browser' from the 'File' menu to close the Path Browser. The application is now removed from the System Interface. The uninstall process WILL NOT delete the application file that was created from the Application Template, or the application variable file that was created from the Variable Template.

B.3.2 Application Interface

In the process of installing an application, a selection is added to the 'Run Application' pull-down menu. Clicking on the application name will open the Application Interface. It is within this window that experiments are created the results displayed either textually or graphically. Figure 3.2 displays an Application Interface.

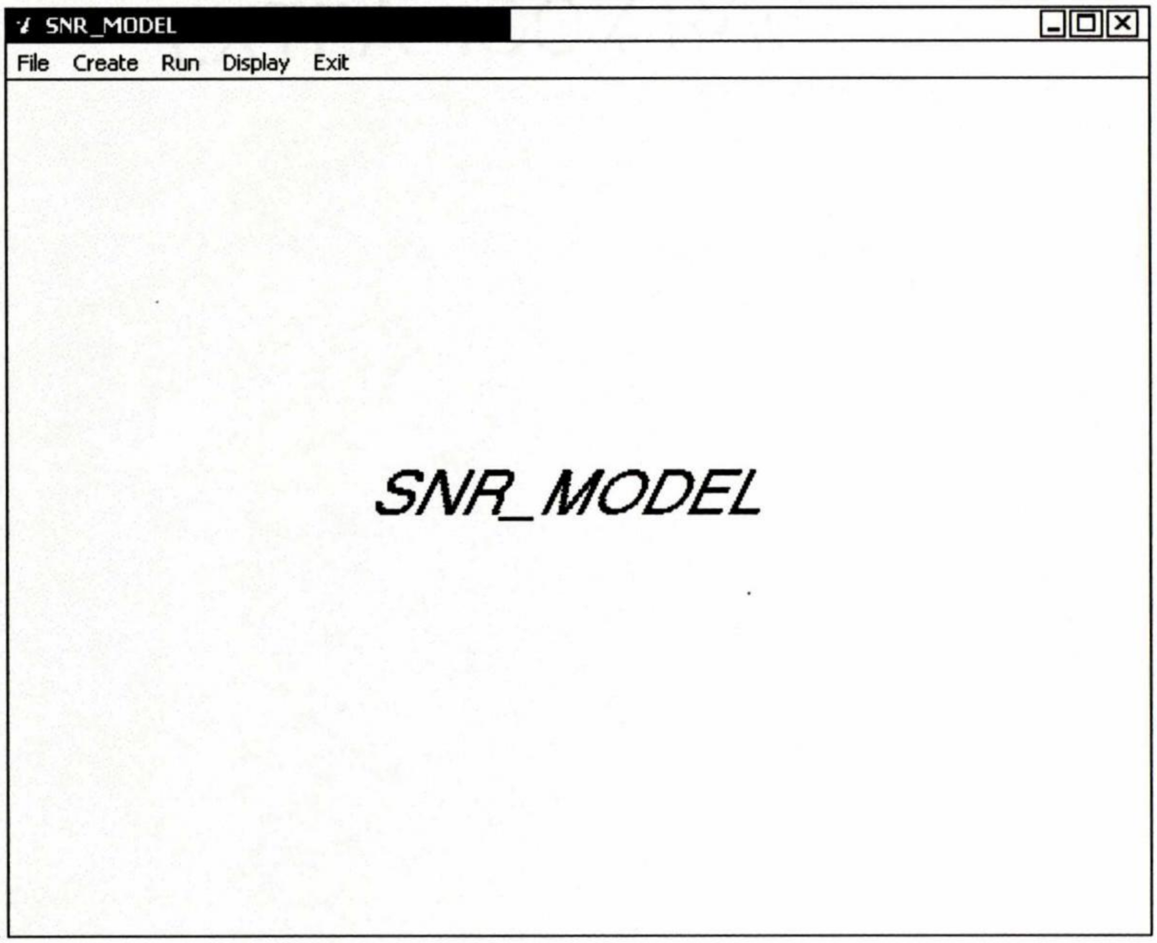


Figure 3.2 Typical Application Interface

When an application is first created, there are of course no experiments associated with the application. Therefore, the user has been restricted to just a few options. These options include the creation of an experiment (2-step process), creating Input and/or Result Interfaces, or exit the application.

B.3.2.1 Creating New Experiments

As was previously mentioned creating an experiment is a 2-step process. The first step involves how the experiment will be developed. There is an interface displayed that gives the user three options of creating the experiment. The second step actually creates the experiment and results object, and updates the Application Interface. The user needs to retype 'GIFTS_MAIN' on the command line to restart the shell.

B.3.2.1.1 From Default Values

Once the user has opened the application window, select 'PreSelect Experiment' from the 'File' pull-down menu. This will open the 'Experiment Select' interface. Figure 3.3 illustrates the 'Experiment Select' interface.

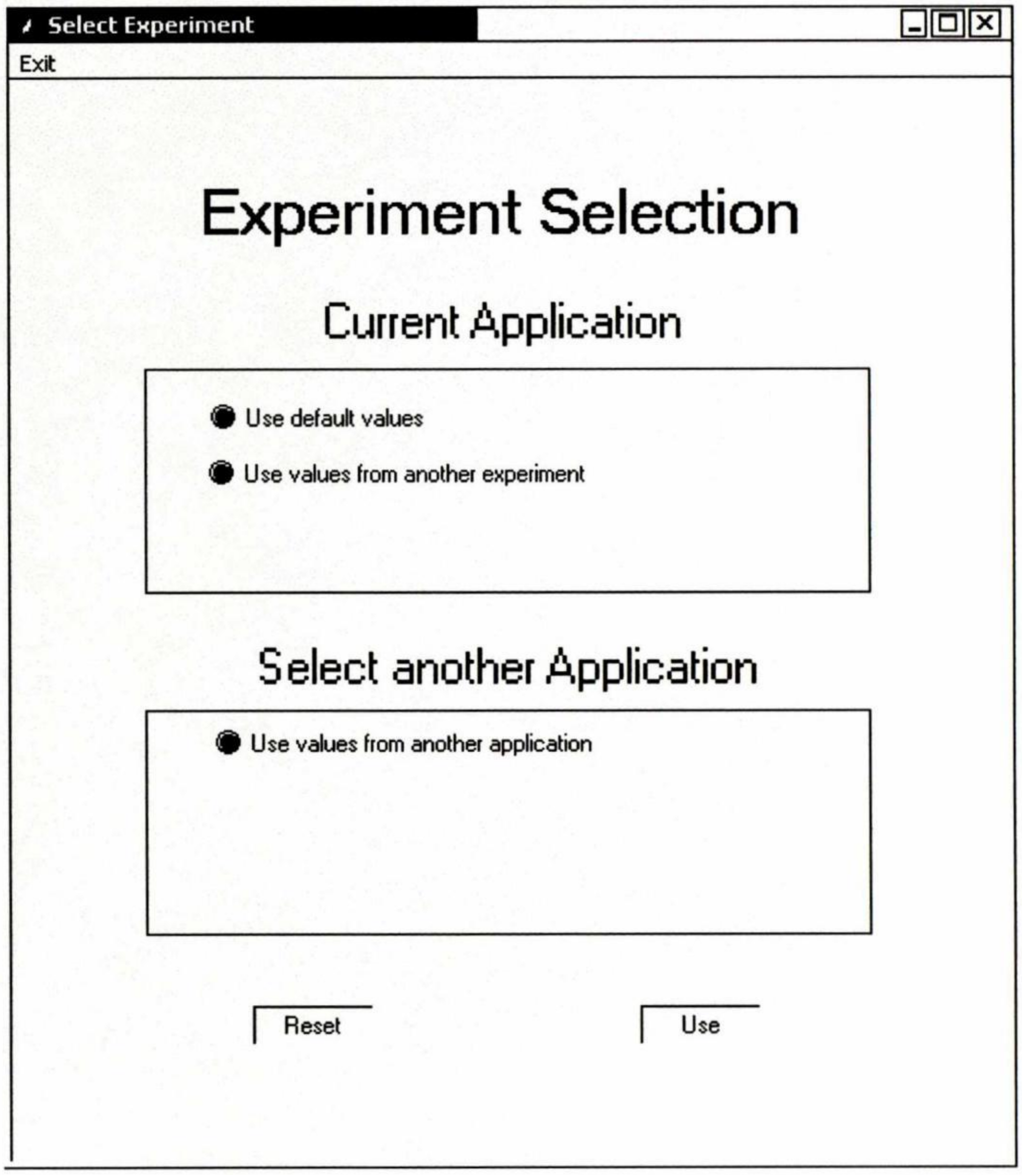

Select Experiment
Exit
Experiment Selection
Current Application
Use default values
Use values from another experiment
Select another Application
Use values from another application
Reset
Use

Figure 3.3 Experiment Selection Interface

Select option 1 (Use Default Values). Then click on the 'Use' pushbutton on the bottom right side of the window. This will bring the user back to the Application Interface. The next step is to select 'Add Experiment' from the 'File' pull-down menu, which has now been activated from the previous step. There will now be a dialog box display prompting the user to provide the path and filename of the external database to use to provide the default settings for the experiment. Enter the information and select 'OK'. Another dialog box opens asking the user to provide a name for this particular experiment. Enter a name for the experiment the click on the 'OK' button. At this point the user will notice that the entire shell has closed. Retype 'GIFTS_MAIN' to restart the shell, than select the application from the 'Run Application' pull-down menu.

B.3.2.1.2 From a Previously Created Experiment

Options two and three from the 'Experiment Select' interface window give the user the ability to use another experiment from the same application, or use another experiment from another application to create a new experiment.

B.3.2.1.2.1 Within The Same Application

The process of creating a new experiment from another experiment starts out the same as for option 1 (Using Default Values). Once the 'Experiment Select' interface window is open, select 'Use Values from another Experiment' option. When this option is selected, a menu will appear with a list of all experiments contained within the current application. Select the appropriate experiment and click on the 'Use' pushbutton. This will once again bring the user back to the Application Interface window. Select 'Add Experiment' from the 'File' pull-down menu. A dialog box will appear prompting the user for a unique experiment name. Enter the information and click on the 'OK' pushbutton. Once again the entire shell will close. Retype 'GIFTS_MAIN' to restart the shell and select the application from 'Run Application' pull-down menu.

B.3.2.1.2.2 From Another Application

In order for this option to be used, there must be at least two applications installed, and the application that the user want to get data from must have at least one experiment contained within that application. Once these two conditions are met, select 'PreSelect Experiment' from the 'File' pull-down menu. When the 'Select Experiment' window opens, select 'Use Values from another Application'. When this option is selected a menu will appear with all applications listed. When the user selects the application of choice, another menu will appear with all experiments listed for that application. Select the experiment of choice. Then click on the 'Use' pushbutton. The user is brought back to the Application Interface. Select 'Add Experiment' from the 'File' pull-down menu. A dialog box is presented to the user to give the new experiment a name. Give an appropriate name and click on 'OK'. At this point the entire shell will close. Retype 'GIFTS_MAIN' or use the up-arrow key to display 'GIFTS_MAIN' on the command line, than press ENTER. Select an application to run.

B.3.2.2 Removing Existing Experiments

When deleting a particular experiment, an application must be in use (Application Window open). From the 'File' pull-down menu select 'Delete an Application'. Select the particular experiment to delete. As in the process of creating an experiment, the shell closes upon experiment deletion. Restart the shell by typing 'GIFTS_MAIN' from the command line, or use the up arrow until 'GIFTS_MAIN' is displayed on the command line, than press ENTER.

B.3.2.3 Selecting an Experiment

In order to select a particular experiment to use within an application, an Application Window must be open. Select 'Select Experiment' from the 'File' pull-down menu. Click on the experiment to use. Once the pull-down menu disappears, the experiment has been selected. At this point the user can run the application using the current experiment selection by selecting 'Run Application' under the 'Run' pull-down menu. After the application runs, a message dialog box will appear informing the user that the application run has completed.

B.3.3 Input/Results Interface Creator Interface

The Input/Results Interface Creator Interface gives the user the ability to create interface windows for both input and results. Figure 3.4 illustrates the Input/Results Interface Creator.

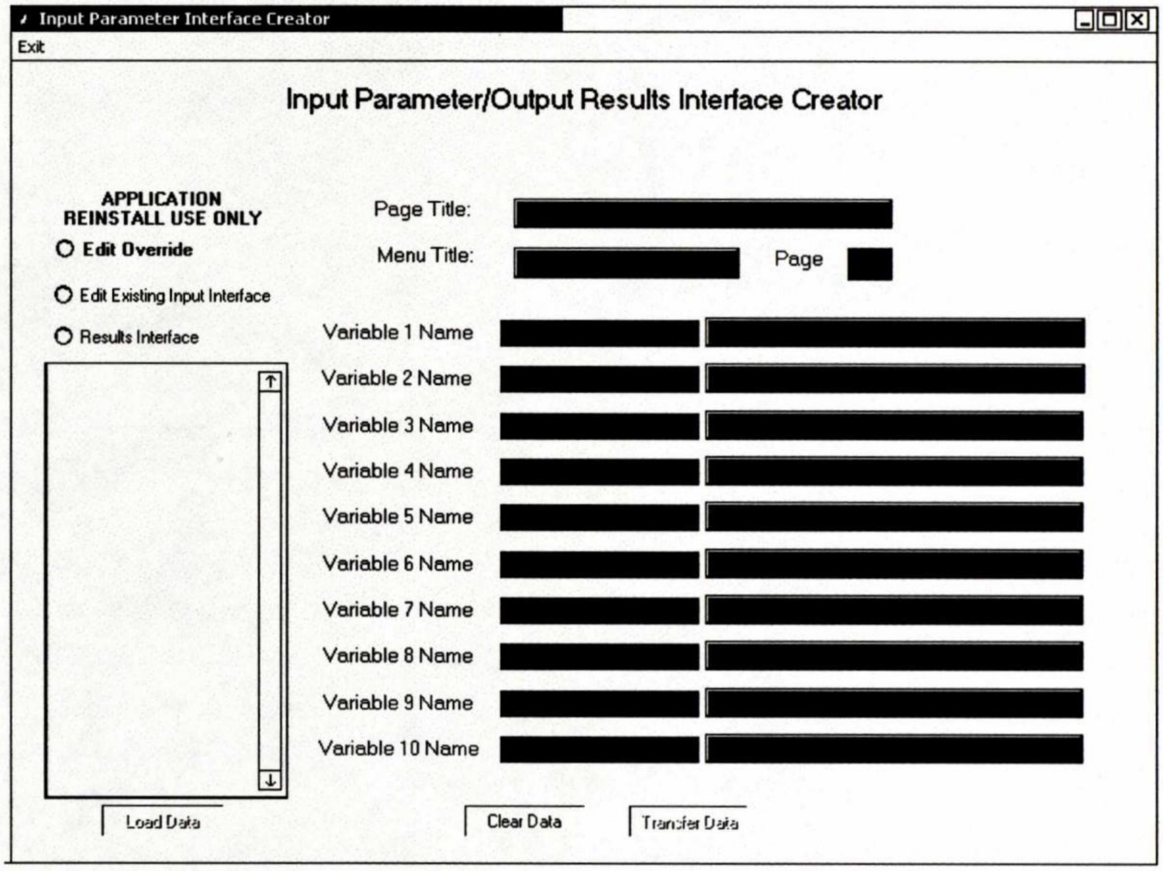


Figure 3.4 Input/Results Interface Creator Interface

The user is given the ability to view and edit input variable(s) to an experiment within a particular application, or view the results of an experiment that has just completed. The interface creator displays the proper variable names given within the application so that a description can be given within the interface window being created. The interface creator also gives the user the capability of reinstalling existing interface windows in the event that the application is uninstalled and reinstalled. The user will have the capability of modifying any existing input or results interface should changes be necessary. The user is limited to creating 20 input and 20 results pages per application. This is to allow the viewing and/or editing of 200 input or 200 results variables.

B.3.3.1 Creating a New Interface

To create a new interface, an Application Window must be open. From the 'Create' pull-down menu select 'Input/Results Interface'. The Input/Results Creator Interface window will appear.

B.3.3.1.1 Input

The default option when the Input/Results Interface window opens is to create an Input Interface. There are two pieces of information that is required before an Input interface is created. There MUST be a Page Title and a Menu Title. The 'Transfer Data' pushbutton will not be enabled until those two pieces of information exist. Once that information is entered, the user than can TAB down to any of the editing boxes and give a proper variable name a descriptive name. Once the information is entered, click on the 'Transfer Data' pushbutton. At this point all windows will close including the shell. Restart the shell by retyping 'GIFTS_MAIN' on the command line or use the up arrow key to display 'GIFTS_MAIN' on the command line, than press ENTER.

B.3.3.1.2 Results

To create a Results Interface, open the Input/Results Interface as described in section 3.3.1. Select the option 'Results Interface'. When this selection is made, a message dialog box will appear informing the user that a Results interface is being created. Click 'OK'. The user will notice that the proper variable names have changed reflecting output result variables that the user defined in the application variable file. As with an Input interface, the user must ensure that Page Title and Menu Title information must be entered before a Results interface can be created. Once this information is available, enter a description for each results variable listed. After the information is entered, select the 'Transfer Data' pushbutton to create the Results interface. All windows including the shell window will close. Retype 'GIFTS_MAIN' or use the up arrow to display 'GIFTS_MAIN' on the command line, than press ENTER.

B.3.3.2 Editing an Existing Interface

There may be a time in which an Input or Results interface may need to be edited. One particular situation is when only a portion of a page is used and the user wishes to add more variables to that page, or the user may want to modify an existing description. For these situations and others, the Input/Results Interface Creator has the capability of modifying existing Input and Results interfaces.

B.3.3.2.1 Input

Follow the directions from section 3.3.1 to open the Input/Results Interface window. With the interface creator in Input mode, select the 'Edit Existing Interface' radio button. A list of all Input Interfaces previously created will be displayed in the list window. Select the window to modify and click on 'Load Data' pushbutton. The selected Input interface window will appear. Bring the Input/Results Interface Creator window to the foreground. Make the appropriate changes to any desired descriptions, than click on 'Transfer Data' to update the Input interface. All windows will close. Restart the shell by typing 'GIFTS_MAIN' at the command line prompt, or use the up arrow until 'GIFTS_MAIN' is shown on the command line, than press ENTER.

B.3.3.2.2 Results

Follow the directions from section 3.3.1 to open the Input/Results Interface window. Set the interface into Results mode by selecting the 'Results Interface' radio button. Next, select 'Edit Existing Interface' radio button. A list of all Results interfaces previously created will be displayed in the list window. Select the window to modify and click on 'Load Data' pushbutton. The selected Results interface window will appear. Bring the Input/Results Interface Creator window to the foreground. Make the appropriate changes to any desired descriptions, than click on 'Transfer Data' to update the Results interface. All windows will close. Restart the shell by typing 'GIFTS_MAIN' at the command line prompt, or use the up arrow until 'GIFTS_MAIN' is shown on the command line, than press ENTER.

B.3.3.3 Reinstalling an Interface

There may be a situation in which an existing application may need to be reinstalled (Uninstall Application and Install Application from the system shell). It could take a great deal of effort on the part of the user to have to recreate all of the previously designed Input and Results interfaces. Therefore, an option has been implemented to reinstall previously created Input and Result interfaces for a reinstalled application. It is at this point that we need to recognize the naming convention for creating Input and Result interface windows.

There are five parts to the naming of either an Input or Results interface window. It is important to understand this naming convention when reinstalling interface windows. We will use as an example the interface window named 'app1IPAGE1'. The five parts are 'app', '1', 'I', 'PAGE', and '1'. Part one is a constant string. It will always be 'app'. The second part refers to the application number. It indicated the number of applications installed. The third part will be either the letter 'I' for Input, or 'R' for Results. The fourth part is the constant string 'PAGE'. The last part is the page number for that particular application. As more interfaces are created, this value will accordingly.

Therefore, 'app1IPAGE1' represents application 1, Input page, and is page 1 of that application.

B.3.3.3.1 Input

To reinstall an input interface, follow the directions in section 3.3.1 to open the Input/Results Interface window. There is a section of the window for reinstalling interfaces. Click on the radio button within this section. A message box will appear informing the user that this selection should only be used for reinstallations. Click 'OK'. Next, select the 'Edit Existing Interface' radio button. A list of all input interface windows will appear within the listbox. Take a look at the page number on the upper right side of Input/Results Creator Interface. This is the page number of that particular application being reinstalled. This is important information that will help the user to select the proper input page to reinstall. Using the naming convention noted in section B.3.3.3. , select the appropriate input interface that matches with the page number within the Interface creator. As an example, if the page number shown on the Input/Creator Interface window is '1', than select the interface that ends with 'PAGE1' from the listbox. Press the 'Load Data' pushbutton. That particular page will be displayed. Within the Input/Results interface, add a name to the 'Menu Name' editbox, and press the TAB key. The user will notice that the 'Transfer Data' pushbutton will become enabled. The user does have the ability to make changes to any of the editboxes at this point, as long as there is information in both the 'Page Name' and 'Menu Name' editboxes. Once the appropriate changes have been made, select the 'Transfer Data' pushbutton. The interface has been reinstalled back into the shell, and all windows will close. Restart the shell by typing 'GIFTS_MAIN' from the command line, or using the up arrow the display 'GIFTS_MAIN' on the command line, than press ENTER.

B.3.3.3.2 Results

Reinstalling a Results interface window is done in much the same manner as an Input interface window, with one difference. Follow directions in section 3.3.1 to open the Input/Results Creator Interface. Select the 'Results Interface' radio button. An information message will appear informing the user that he/she is indeed selecting a Results Interface. Press 'OK'. Than select the 'Edit Existing Interface' radio button. A list of Results Interface windows will appear in the listbox. Then select the 'Reinstall Application' radio button. An information message will appear warning the user of when to use the radio button. Press 'OK'. The selected Results interface window will open. Using the Input/Results Interface window, add in an appropriate name in the 'Menu Name' editbox and press TAB. The 'Transfer Data' will become enabled. The user has the option of making changes to any of the editboxes, or to transfer existing data to the existing interface. Once all changes are made, press the 'Transfer Data' pushbutton. All windows including the shell will close. Retype 'GIFTS_MAIN' from the command line, or use the up arrow to display 'GIFTS_MAIN' on the command line, than press ENTER.

B.3.4 Figure Creator Interface

In order to give the user more flexibility in developing various types of graph charts, GPET-GIFTS has given the user a way of selecting variables to add to a graph, select the type of graph to implement, and plot new data to an existing graph. The user is also given the ability to add title and axis labeling, add grid lines, select normal or semilog graphing parameters in the x- or y- axis directions, as well as adding and modifying legends. Figure 3.5 illustrates the Figure Plotter Interface.

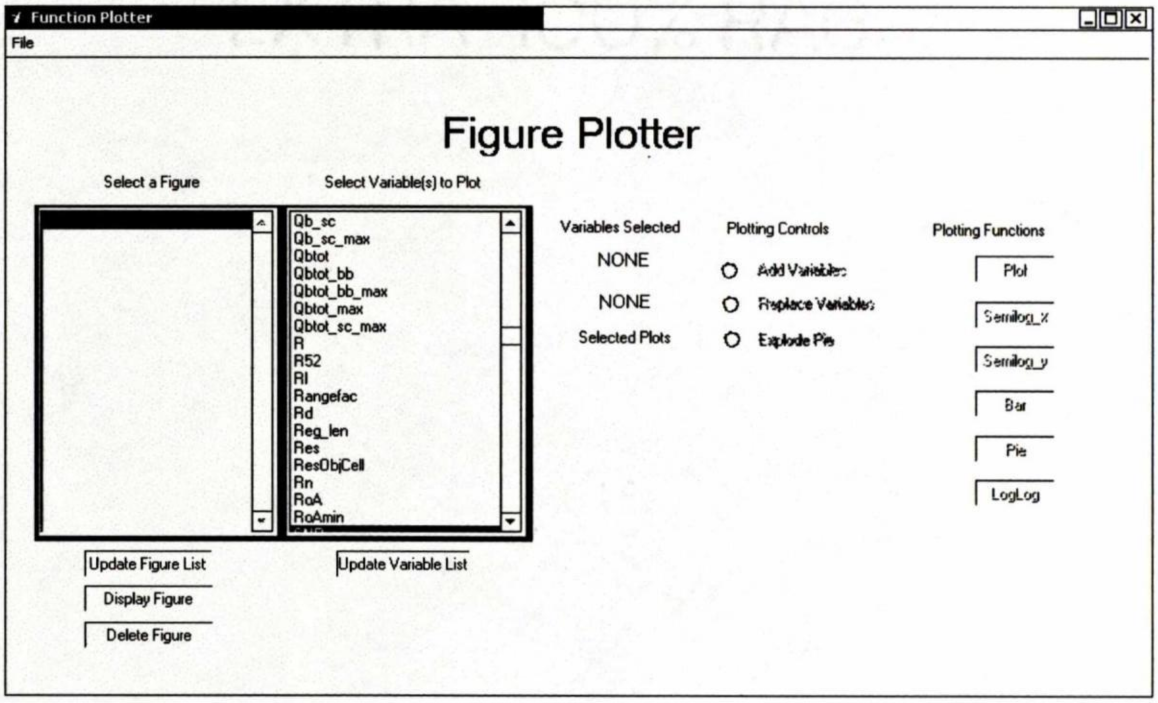


Figure 3.5 Figure Plotter

B.3.4.1 Creating a New Figure

In order to create a new figure, a particular experiment for an application must be selected and ran first so that the variables are available within the local workspace. After running the experiment, open the Figure Plotter Interface by selecting 'Figure' from the 'Display' pull-down menu within the Application Interface.

B.3.4.1.1 Selecting Variables

Within the Figure Plotter is a listbox that displays the all the variables within the local workspace. Next to the listbox is a section named 'Variables Selected'. When a variable is selected, that variable name is inserted into the top box of this section. Also, there is another section named 'Selected Plots'. With one variable selected "Pie Chart Only" will be displayed in this section. In order to select a second variable, the CTRL key will need to be used. Scroll up or down to find the second variable. While holding down the CTRL key, click on the second variable to select it. The 'Variables Selected' section will display the second variable, and the 'Selected Plots' section will display "All Other Plots".

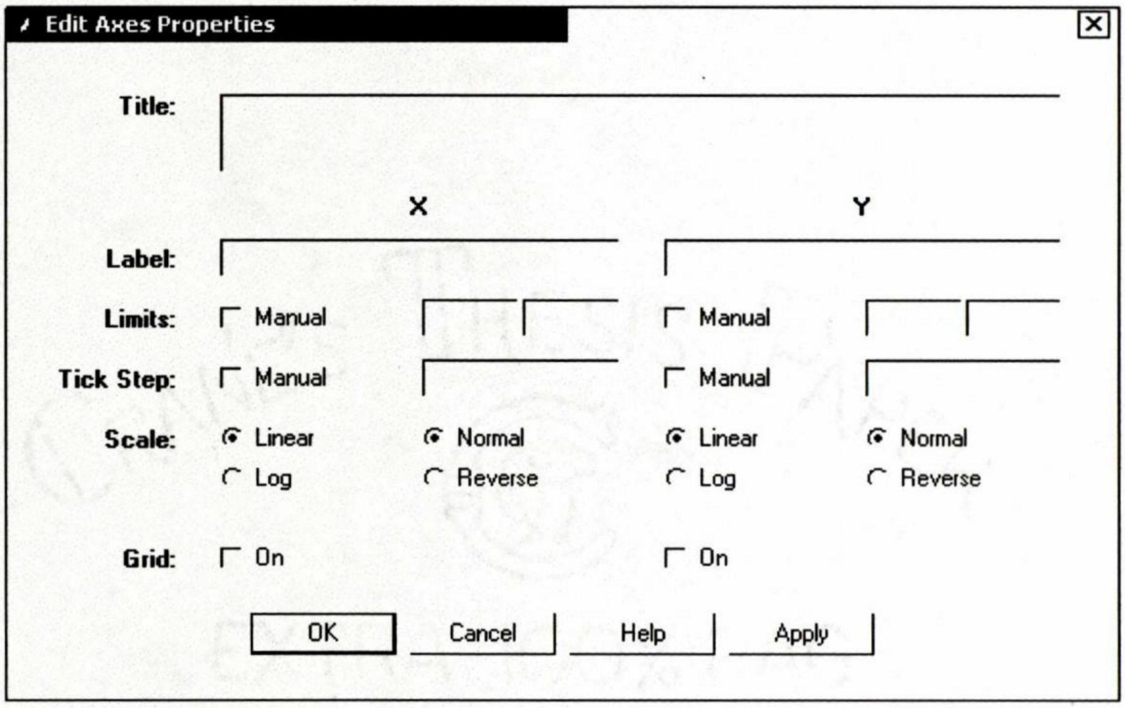
B.3.4.1.2 Graph Type Selection

The Figure Plotter has six available plot types. They are Plot, Semilogx, Semilogy, Bar, Pie, and LogLog. Plot is a standard plot with x and y-axes. Semilogx is a standard plot with the x-axis in log format. Semilogy is a standard plot with the y-axis in log format. A bar chart displays vertical bars. A pie chart is as its name implies, a circular plot separated into pie-wedge shapes. A LogLog plot is a standard plot with both axes in a log format.

The user will notice that with only one variable selected, only the pie chart pushbutton is enabled. With two variables selected all plot types are available except the pie chart.

With the appropriate variable(s) selected, click on the particular plot of interest. A figure window will appear with the appropriate variables plotted. If the user wants to add more lines to this figure, go back to the Function Plotter. Adding more lines should only be done using the plot, semilogx, semilogy, or loglog type plots. It should not be attempted on either a pie or bar chart plot. At the Function Plotter, select two more variables to plot. Than, select 'Plot on Current Graph' radio button within the 'Plotting Controls' section. Then, click on the same plot pushbutton as was done on the first plot. The user should now see two lines within the same figure. Additional lines may be added using the same approach as was previously described.

With the figure displayed, the user has the ability to edit the figure to add such items as a title, x- and y-axis labels, adjust lower and upper limits on both axes, among other adjustments. Figure 3.6 illustrates the 'Edit Axes Properties' dialog box.



	X		Y	
Title:				
Label:				
Limits:	☐ Manual		☐ Manual	
Tick Step:	☐ Manual		☐ Manual	
Scale:	[✓ Linear] Log	[✓ Normal] Reverse	[✓ Linear] Log	[✓ Normal] Reverse
Grid:	☐ On		☐ On	

Figure 3.6 Edit Axes Properties Dialog Box

To open the 'Edit Axes Properties' dialog box click on the 'Tools' pull-down menu on the figure, than select 'Axes Properties'. After all modifications are made to the figure, the user can save the figure. If the user chooses to save the figure, ensure that it is saved to the same directory as the application file(s). If the figure is not in the application directory, the user will not be able to display it at a later time for editing and/or viewing.

B.3.4.2 Editing an Existing Figure

This feature gives the user the ability to add additional graphs, or edit textual portions of a previously saved figure. In order to edit an existing graph, the Figure Plotter Interface window must be open. Refer to section B.3.4.1 to open the Figure Plotter. On the left side of the Figure Plotter Interface is a listbox titled 'Select a Figure'. This listbox displays all previously created figures for a particular application. Highlight the figure to be edited by clicking on it. Than press the 'Display Figure' pushbutton to open the figure. At this point the user can add additional lines by following the procedure in section B.3.4.1.1 (SELECTING VARIABLES). Once the variables are selected, click on the 'Plot on Current Graph' radio button. Select the appropriate plot function that was used on the figure to plot the current line. Make any label changes by using the 'Edit Axes Properties' dialog box. Open the 'Edit Axes Properties' dialog box by following the procedures in the last paragraph of section B.3.4.1.2. Once all changes have been made select 'Save' from the 'File' pull-down menu on the figure to save the figure. Close the figure by selecting 'Close' from the 'File' pull-down menu.

APPENDIX C – VARIABLE TEMPLATE

```
                    %variable_init.m
%
%Created by James M. Mengert
%Modified on: 8/31/2002
%
%This file is used in conjunction with the inputCreator()
%interface window.  It gives the user the ability of
%knowing which variables are being labeled within the
%interface window.
%
%DIRECTIONS TO USE THIS FILE
%Within the switch statement are a number of case
%statements.  Within each case statement are 10 variables
%names Variable1 to Variable10.  These variables at this
%point are blank strings.  The first 200 variables are for
%the Input Interface, and the second 200 variables are for
%the Results Interface.
%
%All the user needs to do is give each variable a string
%representing the variable name that the user created
%within the associated application.  Once the varible
%strings have been placed, save the file to the application
%subdirectory that the user has created.  Refer to the
%user's manaul on creating an application subdirectory.
%
%SPECIFIC STEPS TO CREATING THIS FILE FOR YOUR APPLICATION
%
% 1.  Ensure there is an application directory for your
%application
% 2.  Copy the variable_init.m file to your application
%directory
% 3.  Open the file.
% 4.  Edit the name on the first line to
%     "your_application_name"_init.m
%     where "your_application_name" is the name of your
%     application.
% 5.  Edit the name of the function so that it is identical
%     to the name on the first line.
%
function variable_init ()
%
%Get handles to StaticText displays
%DO NOT EDIT THIS SECTION
%-------------------------------------------------------
StaticTextHandle101 = findobj ('Tag', 'StaticText101');
StaticTextHandle102 = findobj ('Tag', 'StaticText102');
StaticTextHandle103 = findobj ('Tag', 'StaticText103');
```

```
StaticTextHandle104 = findobj ('Tag', 'StaticText104');
StaticTextHandle105 = findobj ('Tag', 'StaticText105');
StaticTextHandle106 = findobj ('Tag', 'StaticText106');
StaticTextHandle107 = findobj ('Tag', 'StaticText107');
StaticTextHandle108 = findobj ('Tag', 'StaticText108');
StaticTextHandle109 = findobj ('Tag', 'StaticText109');
StaticTextHandle110 = findobj ('Tag', 'StaticText110');
%
StaticTextHandle50 = findobj ('Tag', 'StaticText50');
page = get (StaticTextHandle50, 'String');
page = str2num (page);
%
RadioButton3Handle = findobj ('Tag', 'Radiobutton3');
ButtonValue = get (RadioButton3Handle, 'value');
%--------------------------------------------------------
%Input variables
if ButtonValue == 0
   switch page
   case 1
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 2
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 3
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 4
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 5
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 6
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 7
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
   Variable10 = '';
case 8
   Variable1 = '';
   Variable2 = '';
   Variable3 = '';
   Variable4 = '';
   Variable5 = '';
   Variable6 = '';
   Variable7 = '';
   Variable8 = '';
   Variable9 = '';
   Variable10 = '';
case 9
   Variable1 = '';
   Variable2 = '';
   Variable3 = '';
   Variable4 = '';
   Variable5 = '';
   Variable6 = '';
   Variable7 = '';
   Variable8 = '';
   Variable9 = '';
   Variable10 = '';
case 10
   Variable1 = '';
   Variable2 = '';
   Variable3 = '';
   Variable4 = '';
   Variable5 = '';
   Variable6 = '';
   Variable7 = '';
   Variable8 = '';
   Variable9 = '';
   Variable10 = '';
case 11
   Variable1 = '';
   Variable2 = '';
   Variable3 = '';
   Variable4 = '';
   Variable5 = '';
   Variable6 = '';
   Variable7 = '';
   Variable8 = '';
   Variable9 = '';
   Variable10 = '';
case 12
   Variable1 = '';
```

```
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 13
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 14
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 15
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';

    case 16
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 17
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 18
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 19
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 20
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
```

```
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   end %End of switch
end %End of IF
%
if ButtonValue == 1
%Result variables
   switch page
   case 1
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 2
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 3
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 4
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
```

```
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 5
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 6
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 7
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 8
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
```

```
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 9
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 10
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 11
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
      Variable10 = '';
   case 12
      Variable1 = '';
      Variable2 = '';
      Variable3 = '';
      Variable4 = '';
      Variable5 = '';
      Variable6 = '';
      Variable7 = '';
      Variable8 = '';
      Variable9 = '';
   Variable10 = '';
case 13
   Variable1 = '';
   Variable2 = '';
   Variable3 = '';
   Variable4 = '';
   Variable5 = '';
   Variable6 = '';
   Variable7 = '';
   Variable8 = '';
   Variable9 = '';
   Variable10 = '';
case 14
   Variable1 = '';
   Variable2 = '';
   Variable3 = '';
   Variable4 = '';
   Variable5 = '';
   Variable6 = '';
   Variable7 = '';
   Variable8 = '';
   Variable9 = '';
   Variable10 = '';
case 15
   Variable1 = '';
   Variable2 = '';
   Variable3 = '';
   Variable4 = '';
   Variable5 = '';
   Variable6 = '';
   Variable7 = '';
   Variable8 = '';
   Variable9 = '';
   Variable10 = '';
case 16
   Variable1 = '';
   Variable2 = '';
   Variable3 = '';
   Variable4 = '';
   Variable5 = '';
   Variable6 = '';
   Variable7 = '';
   Variable8 = '';
   Variable9 = '';
   Variable10 = '';
case 17
   Variable1 = '';
```

```
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 18
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 19
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    case 20
        Variable1 = '';
        Variable2 = '';
        Variable3 = '';
        Variable4 = '';
        Variable5 = '';
        Variable6 = '';
        Variable7 = '';
        Variable8 = '';
        Variable9 = '';
        Variable10 = '';
    end %End of switch
end %End of IF
%
```

```
%DO NOT EDIT THIS SECTION
%-----------------------------------------------
set (StaticTextHandle101, 'String', Variable1);
set (StaticTextHandle102, 'String', Variable2);
set (StaticTextHandle103, 'String', Variable3);
set (StaticTextHandle104, 'String', Variable4);
set (StaticTextHandle105, 'String', Variable5);
set (StaticTextHandle106, 'String', Variable6);
set (StaticTextHandle107, 'String', Variable7);
set (StaticTextHandle108, 'String', Variable8);
set (StaticTextHandle109, 'String', Variable9);
set (StaticTextHandle110, 'String', Variable10);
%-----------------------------------------------
```

VITA

Jim Mengert was born in Upon graduating from high school in June 1976, he enlisted in the military, and remained there until his retirement in June 1996. Jim's academic accomplishments include receiving his Associate of Science degree in Computer Science from Tidewater Community College, Virginia Beach, VA, in December 1997. He received his Bachelor of Science Degree in Computer Engineering with a minor in Computer Science from Old Dominion University, Norfolk, Virginia, in May 2001. He will receive his Master of Science Degree in Computer Engineering from Old Dominion University in May 2004. Jim is currently employed as an Engineer within the Tactical Communication Support branch with the Space and Naval Warfare Systems Center (SPAWAR).