TOWARD THE DEVELOPMENT OF CONTROL SOFTWARE FOR AN OPERATOR
INTERFACE IN THE DISTRIBUTED AUTOMATION ENVIRONMENT

By

Usha Jayaraman

Project submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Engineering
in
Industrial and Systems Engineering

APPROVED:

Michael Deisenroth
Dr. M. P. Deisenroth, Chairman

Dr. O. Eyada

Sankar Jayaram

April 30, 1992
Blacksburg, Virginia
TOWARD THE DEVELOPMENT OF CONTROL SOFTWARE FOR AN OPERATOR INTERFACE IN THE DISTRIBUTED AUTOMATION ENVIRONMENT

By
Usha Jayaraman
Dr. M. P. Deisenroth, Chairman
Industrial and Systems Engineering

(ABSTRACT)

Advances in technology have led to fully integrated operator consoles which offer a broad range of solutions. Industry's move towards networking has resulted in the operator consoles being located at critical sites throughout the plant-floor. However, a lot of programming effort is required in order to use these consoles effectively.

The project addressed the problem by developing a software interface for the future user in a distributed automation environment. The system developed is capable of operating on IBM industrial computers, connected through a communications network on a DAE environment. By using the functions provided, the future user is insulated from the lower level control sequences of the operator interface and need only be concerned about the operation of these functions.
A description of the development of each phase of the project is given, and the purpose and operation of all subroutines are also presented in this report.
ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my advisor, Dr. Micheal P. Deisenroth, for his guidance and support in this effort, especially in defining the project topic.

I would also like to thank Dr. O. Eyada and Dr. S. Jayaram for serving on my project committee.

Finally I wish to thank my husband, Radha, for his relentless support and encouragement.
# TABLE OF CONTENTS

1.0 CHAPTER 1. INTRODUCTION ...........................................1  
   1.1 Background .....................................................1  
   1.2 Problem Statement .............................................5  
   1.3 Project Objective and Approach ................................7  
   1.4 Documents Available ..........................................11  

2.0 CHAPTER 2. LITERATURE REVIEW .................................13  
   2.1 The History of Operator Interfaces ............................13  
   2.2 Advances in Operator Interface Technology ..................15  
   2.3 Future Trends ..................................................19  
   2.4 Research Project ..............................................20  
   2.5 The Project and Graphics ......................................22  

3.0 CHAPTER 3. DAE ENVIRONMENT ....................................24  
   3.1 Application development in the DAE Environment ..........24  
   3.2 DAE and the Plant-Floor Environment .......................27  
   3.3 DAE and VPI MARL .............................................28  
   3.4 Basic Concepts of DAE .......................................29  

4.0 CHAPTER 4. GE OPERATOR INTERFACE TERMINAL .................34  
   4.1 GE Operator Interface Terminal Overview ....................34  
   4.2 Operation and Functionality ..................................38  
      4.2.1 Special Keys ............................................38
LIST OF REFERENCES .................................................. 94

APPENDICES ............................................................. 98

APPENDIX A. GRAPHIC CHARACTER SET 1 ......................... 98
APPENDIX B. GRAPHIC CHARACTER SET 2 ......................... 99
APPENDIX C. QUAD SIZE CHARACTER SET ......................... 100
APPENDIX D. RDS FIELDS ............................................. 101
APPENDIX E. OIT FIGURE ............................................ 103
APPENDIX F. USER'S GUIDE ......................................... 104
# LIST OF ILLUSTRATIONS

3.1 DAE Systems Enabler ........................................26
5.1 Schematic Description of Network Configuration ...46
6.1 Node-to-Node Communication .................................54
6.2 Layout of Request Work Area .................................56
6.3 Control Blocks on Node1 ....................................59
6.4 Control Blocks on Node2 ....................................60
6.5 Functions Available in each Control Group ..........65
6.6 Functions in I/O Control Module .........................77
6.7 Main Menu Screen ........................................80
6.8 Configuration Menu Screen .................................81
6.9 Screen Operations Menu Screen .........................83
6.10 Functions Used for Programming Screen 1 ............85
6.11 Functions Used in Receiving Sample Data Option ...86
6.12 Graphics Display Menu Screen .........................87
6.13 Functions Used in Bar Graph Display ..................88
6.14 Functions Used in Fl. Manf. Cell Layout Option ......89
6.15 Function Key Operations Menu Screen .................90
CHAPTER 1. INTRODUCTION

1.1 Background

The industrial plant-floor console, which typically features a computer driven graphics display has for sometime been commonly termed the operator interface. The plant-floor console functions as the operator's window to the world. In recent years, new terms such as operator stations, smart terminal and work stations have evolved. Regardless of the name given to it, the plant-floor console serves an important purpose, and that is to aid the plant-floor operator in his or her operations and make him or her more effective. In the case of recent plants, new operator interfaces have been introduced which help in saving manpower and lessen the worker's load and aid the operator to grasp the process conditions more comprehensively and easily.

Wickens[1] has described process control as an area in which the issues of decision making, perception, and memory take on importance. Today, plant-floor operators are faced with a high amount of monitoring activity and relatively occasional intervention in plant processes. One of the early, yet definitive, works on the nature of the operator in process control was that of Coombs[2], who noted that the
emphasis in the process control operator's job was shifting toward the ability to take in information, organize it, and interpret it for action. The goal of the operator, he further determined, was fivefold: 1. to regulate or stabilize the process; 2. to adjust the process in order to optimize; 3. to make changes from one product to another; 4. to avoid breakdowns; and 5. after breakdown, to regain control of the process as soon as possible. The process operator's goals and skills identified by Coombs have relevance for the modern process control room.

In a typical plant environment, operator interfaces are often linked to a host computer or a programmable controller using a serial communications interface. Gardner[3] has described terminals which have no resident memory as "dumb" terminals. These, he further points out, derive their intelligence from the host computer via command sequences over a communication link. The growth of intelligent operator interface devices has led to real-time functionality and relieves the controller of any burden specific to the interface. They may possess alarm management, database management, and other logic and math capabilities.

Alternative technologies for operator interfaces have evolved into a burgeoning industry during the past decade.
Due to the proliferation of the operator interface devices, a system classification is now being adopted by several manufacturers and their customers[3]. This system organizes the various types of operator interfaces into five different distinct levels that compare relative size, technology and complexity:

1. Single function devices,
2. Alphanumeric displays,
3. Video control panels,
4. DOS-based workstations, and
5. Non-DOS based cell or area level workstations.

In the above classification, the operator interface devices belonging to the first category are the simplest in terms of functionality and operation. Conventional pushbuttons, pilot lights, thumbwheel switches etc., fall in this category. The operator interface devices in the second level include general purpose devices like CRTs, matrices of LEDs or LCDs. These typically present information supplied from an outside source and sometimes possess operator input capability. The devices in the third level are designed to replace the operator interfaces belonging to the first two levels. They offer real-time functionality in a single package, and may include alarm management, logic, and math capabilities. Finally, the devices in the fourth and fifth category distinguish themselves from the previous levels by
their capabilities for pixel addressable graphics, database management, disk-based memory media, etc. They offer color graphic capability, communication drivers for connection to programmable controllers, and also may have the ability to accept user written high-level language programs.

Another major change taking place in today's manufacturing environment is the introduction of networking. The communication network makes it possible to access information located on remote control systems and provides a means of routing messages and data between the various plant-floor devices and controllers. The need for systems that can share information is being recognized, and thus plant-wide information systems or distributed control systems are now emerging.

These distributed control systems attempt to integrate the "islands of information" into a single expandable system. With the advent of distributed control systems, operator consoles are getting distributed throughout the plant-floor. Plant-wide integration is now becoming possible with the emergence of the distributed processing systems linking plant-floor equipment. These systems are now available as commercial software products and attempt to provide solutions for Computer Integrated Manufacturing (CIM). These products facilitate integration through a flexible
systems architecture with uniform interfaces to all devices. Also, these products offer plant-floor support for managing data and communications with devices on local and remote nodes of the network. BASEstar by DEC, Sockets by HP, and Distributed Automation Edition (DAE) by IBM are some of the commercially available integrated factory-floor software.

1.2 Problem Statement

The operator sitting on the plant-floor needs a means by which he or she can communicate effectively with an industrial machine or control equipment. As already pointed out, currently there exist a vast number of operator interfaces. These devices provide a buffer between the operator and the machine or the equipment that needs to be controlled or monitored.

Process status information is made available to the plant-floor operator, thus aiding him or her in monitoring the process. Plant-floor consoles could be used for data entry purposes where the data can be control data or simple data required by the machine. Additionally, information and commands received from a "high-level controller" can be captured by the controller connected to the console, and displayed on the display screen of the plant-floor console.
Interfacing with a plant-floor console requires a communication link between the controller or the host computer and the operator interface. This connection between the corresponding ports of the controller and the operator interface is usually a serial communication link at an RS-232-C or an RS-422 signal level. In order for the communication to be successful, special control programs or control modules are required. These control programs reside in the controller or the host computer and are executed every time any communication between the controller and the operator interface is necessary.

Operator interfaces recognize distinct character sequences or formats for displaying graphics characters, editing, configuring, performing screen storage functions, etc. In order to use the operator interface effectively, familiarity with the above formats is required. These control programs take care of the opening and the closing of the ports on the controller, and are also responsible for sending and receiving data in the appropriate format between the operator interface and the host computer. These interface modules, then, can be considered the most critical part of the interfacing procedure. The successful operation of the operator interface depends greatly on the effectiveness of these modules. Thus application developers spend a lot of time in creating these control modules.
To reduce the burden of software development associated with operator interfaces, a system is needed which will allow the applications programmer to communicate easily to the available operator interface. This "communications layer" will allow the applications programmer to interface with the plant-floor console without bothering about character formats or sequences, opening ports, closing ports, etc. Additionally, this "communications layer" would permit an operator interface located on the communications network to display and receive data from multiple sources. Thus, the system should be able to route the data from a source node to the appropriate controller connected to the operator interface and similarly, it should be able to receive data from the operator interface via the controller and route it to a desired destination.

1.3 Project Objective and Approach

The intent of this research encompassed the development of control software for an operator interface terminal which will operate in the Distributed Automation Edition (DAE) environment on IBM industrial computers, connected through a communications network. The system developed is capable of performing communication between any computer (on the communications network) and the GE operator interface
terminal (GE OIT) and encompasses all the programmable functional capabilities of the GE OIT.

The Manufacturing, Automation and Robotics Laboratory in the Department of Industrial Systems Engineering of Virginia Polytechnic Institute and State University was planned to provide a facility for instruction and research in the integration and control aspects of a computer-based manufacturing environment. DAE is used as a software development platform in the laboratory to produce the required system-wide integration between the various controllers - system controller and the work-cell controller. The control software was developed to enable the future programmer to use the developed system with the large number of GE OITs available, without having to go through extensive programming efforts. Also, the software development was done in the DAE environment to enable communications between the different nodes of the network and the GE OITs located at the various work-cell controllers. Thus a node acting as a system controller can send information to the GE OIT, which may be connected to a node acting as a work-cell controller, and similarly, the work-cell controller can route the information received from the GE OIT to a desired destination in the network.
To accomplish the above purpose, the system development was done in three phases:

1. Development of library routines,
2. Development of the I/O control module for the control node, and
3. Development of a demonstration program.

In the first phase, all the functional capabilities of the GE OIT were developed, and then integrated to form a library of routines which can be used repetitively in the development of control software by an applications programmer. In developing the routines, it was considered important that they be practical and easy to use. Thus, they have simple parameter settings and can be easily called by a future programmer who needs to interface with the GE OIT.

The library functions developed in the first phase provide different classes of software control for the GE OIT, such as character attribute control, line attribute control, configuration control, etc. These functions are best suited for the control they are meant to provide, and the applications programmer can use the functions, without going through large programming efforts, thereby reducing the time for software development.
The second phase involved the development of an I/O control module for the GE operator interface terminal. This module directly interfaced with the GE OIT and provided a link between the library of routines and the GE OIT. This module was responsible for routing messages between the GE operator interface terminal and any application program located on any node of the Token-Ring communications network. Thus, a programmer can create applications, which use the routines developed in the first phase, and communicate with the GE OIT from any node of the network.

Apart from the tasks described above, the I/O control module also performed the required initialization and parameter setting of the serial port on the host computer in order for the serial communications link between the host computer and the GE OIT to be effective.

The third and final phase of the project involved the development of a demonstration program. This program serves as an example of how the routines developed in the first phase of the project can be used. Additionally, it provides sample code for an applications programmer attempting to interface with the GE OIT, and helps in emphasizing the communications flexibility of the system.
1.4 Documents Available

The following documents need to be reviewed in order to gain a full understanding of the system developed:

1. Project Report: This document contains a high level summary of the research. It explains the methodology of approach and explains the three phases of system development. It also outlines the basic concepts of DAE and the operation of the GE OIT.

2. User's Guide: This document explains the two major modules of the system in detail: the library of routines and the I/O control module. It also contains a brief description of the demonstration program. The library functions, their parameters, and return codes are explained, and the operation of the functions constituting the I/O control module are also given.

3. GE OIT User's Manual: This manual provides a description of the hardware and programmable features of the GE OIT. The user can refer to this manual for any details regarding the GE OIT.

4. IBM, Application Programming Volume I, Communications System/2: It provides application
designers and programmers with the information needed to begin writing applications in the DAE environment. The basic system concepts and terminology of DAE are introduced here.

5. IBM, Application Programming Volume II, Communications System/2: It provides the programmers with details about the different API calls and request data structures that can be used in an application program.
CHAPTER 2. LITERATURE REVIEW

2.1 The History of Operator Interfaces

Dumont[5] has defined an operator interface as a device or system that facilitates the interaction of a human operator and an intelligent control system to accomplish a desired result. The goal of an operator interface device is to foster an intelligent monitoring and management of an industrial process.

Operator interfaces serve numerous purposes in an industrial environment. They can be used for data entry where data could be process control data or simple data input required by the machine. They prove useful in situations where it is essential to warn the operator of a malfunction of any equipment or a hazardous situation[6]. Status information is made available to the operator through the plant console, which is useful for monitoring a control system. Thus the operator can make sure that events are occurring as they should and can attempt to monitor and, if required, control the process.

Tuttle[7] has described four generic functions that are performed at the operator console:
1. Monitoring the Process - Display real-time and historical data in a variety of formats.
2. Decision Support - The operator ensures that the events are occurring as they should, and determines the course of action, given a set of circumstances.
3. Supervisory Control - Once the course of action is determined, the appropriate devices are actuated, and parameters are reset or modified.
4. General Administration - Develop and evaluate the alternatives, generate reports and schedules, etc.

The initial operator interface devices included pilot lights, push buttons and switches which were typically hard-wired to the machine or the process being controlled. Their primary purpose was to convey information to and from the operator. Then came the CRT console and other display media which were capable of presenting information in a variety of formats. These devices had no resident memory and were driven via command sequences over a communications link.

As industrial control systems became more complex, the need for easy-to-use interfaces for these systems became ever more pressing. Manufacturers were constantly alert for ways to make plant-floor operation more productive and profitable, and the operator interface served as a logical area of concentration where improvements in technology
would have a direct impact on the successful operation of a manufacturing process.

2.2 Advances in Operator Interface Technology

According to Dumont[5], the fundamental basis of all improvements in operator interface is the conviction that the operator is the one person who has the most impact on the success of a particular operation. Advances in the operator interface technology has led to fully integrated operator interfaces, which offer a broad range of solutions. These operator interfaces reduce integration time and down time, are easy to use, and can withstand demanding industrial environments. These advances contribute in general or specific ways to the operator’s need for more information.

With the introduction of network based control systems, access to process control data became easy, and terminals were installed at critical sites throughout the plant. Networks provided access to decision making applications, tools, and information on remote control systems in the plant. Coordination with different control systems became possible and remote data could now be accessed easily. Different types of control system equipment and terminals got linked to a communications network. Operator consoles anywhere on the network could now be reconfigured, and
information could be easily displayed and retrieved from their display screens. Plant floor machines could now communicate with each other easily, making it easier to download, upload programs from or to control equipment, machines, etc. Today's distributed control systems with color graphics monitors are replacing the yesterday's "semigraphic" panels. Today operator interface consist of one or more monitors in place of the numerous indicators and panels of the past. In addition to the advantages of being more interactive, instructive, and informative, the new consoles require significantly less space.

CRTs, matrices of LEDs or LCDs, and some of the newer flat-screen technologies are today's popular display media[3]. These devices are capable of displaying information in a variety of formats and are also capable of graphics, either in monochrome or multiple colors, depending on the display medium. They may have memory ranging from 2,000 bytes to over 65,000 bytes. Glow[10] has predicted that flat panel displays will become more popular in industrial plants in the form of operator stations and programmable displays. Flat panel displays are inherently compact and rugged, and more suited (than CRTs) to withstand the harsh industrial environment.
In the last decade, touch screens have emerged as a proven input device for industrial applications. It is increasingly becoming the preferred human interface for many automated and computer based applications[11]. They eliminate the need for keyboards and extensive operator training. The greatest advantage of touch screen technology is that it gives easy access to important information with a simple touch of finger. According to Petersen[12], factory automation and machine control have become a popular application for touch screens.

Other advancements in operator interface technology include Video Control Panels. Introduced in 1986, they offer real-time functionality to the user. Operator input is by membrane buttons mounted around the display screen. They relieve the operator of any burden specific to the interface and, sometimes may include alarm management, logic and math capabilities.

The term workstations has lately come into popular use. It is borrowed from design engineers and draftsmen who have long used it to refer to their "computerized drawing boards". Workstations provide pixel addressable graphics, have high resolutions, thousands of colors, fonts, shading, database management capabilities and disk-based memory media.
Literature discussing the control software developments for operator interfaces are currently unavailable. Any research that has been conducted, has been done by commercial organizations, and any literature that exists is proprietary. There is, however, some available in marketing literature from a few companies which provide an indication of the extent to which the technology has progressed.

Almost all of the so called dumb terminals are programmed by the software developer. Many hours of programming effort are required in order to use these operator interfaces effectively. Workstations, however, are more difficult to program. Most control and instrumentation engineers don’t want to learn sophisticated programming techniques; they especially don’t want to have to learn difficult programming techniques for each piece of control equipment, whether from a different manufacturer or a single manufacturer[13]. Thus, the newer video control panels and workstations are accompanied by communication drivers. The software accompanying these operator interfaces allow the user to configure and perform operations on them, without additional programming expertise. In some cases, the user has the option of purchasing the control software. Other integrated software packages, which are oriented towards the more sophisticated workstations, offer color graphics capabilities and communication driver for connection. Some
of the operator interface software packages available today, include the FIX from Intellution, FactoryLink from USData, ONSPEC from Heuristics, and many others.

According to Beum[14], a new cost-effective industrial CRT, that would serve as a primary operator station, with more flexibility and functionality than traditional hard-wired control panels, but without the need for programming expertise is now being introduced. The PanelView is such a terminal. It runs on a PanelBuilder menu-driven software package which runs on an IBM-PC, and can be connected via an RS-232-C cable to the terminal.

2.3 Future trends

The single highest cost component in industrial color graphics devices is the CRT, whose costs have risen significantly over the last few years. The evolution of flat panel displays towards more functionality will have a major impact on the cost and physical size of industrial operator interfaces. Workstations and windowing environments can improve the user interaction considerably. These will facilitate the much needed plant-wide collaboration among control system operators.

There is a growing trend towards closer coupling of interface devices to the controller or host computer.
Remote I/O links are beginning to appear and back plane connections of interface to controller are not far behind. According to Dumont[5], the next generation of operator interfaces will be oriented towards information retrieval. Thus, operators using these interfaces will be better informed about their processes and will be able to initiate decisions to increase output and efficiency.

What new technical terms will be used to name the operator interface is anyone's guess. What seems clear though, is that plant-floor control consoles will continue to become more like workstations, while remaining in a important sense, operator interfaces.

2.4 Research Project

The focus of the project was to develop a system which will provide a software interface to be used for receiving and sending information between the GE OIT and any node of the communications network. All the programmable features of the GE OIT were developed to form a library of functions, which can be used repetitively in application development. As already mentioned, the DAE environment was used for developmental purposes because it provides the required system-wide integration. Thus the system will permit a source node to relay messages to the GE OIT and also route the data received from the GE OIT to a requesting node. The
system will aid a future user to interface with the GE OIT in a DAE environment.

The GE OIT is a 12" CRT terminal designed for use in a harsh industrial environment. It is designed to support a wide variety of existing software, while focusing on the requirements and applications of the plant-floor. It has 32k bytes of resident memory and supports limited graphics capability. It supports a variety of programmable character attributes, screen and function key programming, clock and data functions, editing features, various configuration modes, etc. In the classification provided by Gardner[3] for operator interfaces, the GE OIT can be categorized as a level two device.

The DAE environment simplifies the implementation of integrated manufacturing applications in a distributed environment. DAE is a "systems enabler" and consists of an application programming interface (API) and a services layer. The API and the services layer together provide a software platform for an application programmer. This software platform forms a base on which the programmer can develop applications and serves to insulate him or her from the complexities of performing house-keeping tasks. Also, the DAE provides basic services that aid the developer to
efficiently manage the resources found in a manufacturing environment.

2.5 The Project and Graphics

The objective of the project is to develop support routines to be used in conjunction with the GE OIT. A portion of this task was oriented towards developing its graphics capability, and providing a future user of the system with the ability to create effective displays on the GE OIT screen. Currently, there exist a wide variety of packages which provide advanced graphics capability to a user.

One set of such graphics packages are directed towards performing graphics display on PC-based computer monitors. Currently, there is an extensive variety of IBM Personal Computers, PCs, XTs, ATs and clones available. Most of the monitors used with the above systems are capable of producing extensive graphics display, which has led to several graphics standards like Color/Graphics Adapter (CGA), the Hercules Graphics Card (HGA), the Enhanced Graphics Adapter (EGA), the Video Graphics Array (VGA), etc.

The routines available with the above software packages provide details on the use of the CGA, HGA, EGA, VGA and the ROM BIOS video services. They provide basic graphics tools and programming detail which can only be used with these special graphic interfaces.
The second set of graphics standards which are oriented towards the more powerful workstations and mainframes are GKS, PHIGS, and PHIGS+. Graphics Kernel System (GKS) was introduced in 1985 as a ISO 2-D graphics programming standard. In 1988, Programmer's Hierarchical Interactive Graphics System (PHIGS) was accepted as the ANSI and ISO standard for 3-D graphics programming. PHIGS includes most of the 3-D graphics primitives and graphical input devices for the developmental of an interactive graphical user interface. This standard is proving useful in the creation of custom CAD software.

This project is specifically directed towards the development of the programmable features of the GE OIT in a DAE environment. About 10-15 percent of the code generated was oriented towards graphics programming. However, this code dealt with programming special escape sequences and character formats which can only be used with the GE OIT. Current graphical standards, such as those discussed above, were not applicable for this project.
CHAPTER 3. DAE ENVIRONMENT

Today's industries are being challenged to compete more effectively in world markets. The introduction of computer applications on the plant-floor has produced cost-effective methods of meeting this challenge. New computer integrated manufacturing (CIM) architectures are making possible the long-held objective: tying factory floor machines to manufacturing systems[4]. The long range goals of process control and the objectives of CIM are now being realized with the advent of distributed control systems and real-time database connections. A wide variety of vendor supplied software packages are now available to meet the control needs of most users.

3.1 Application Development in the DAE Environment

As already mentioned, three companies that dominate the installed bases of enterprise-wide CIM are IBM, HP and DEC[4]. The Distributed Automation Edition (DAE) or the Communications System, is one of the key pieces of IBM's CIM architecture. It is defined as a "systems enabler", consisting of an application programming interface (API) and a services layer. A systems enabler can be defined as a set of preprogrammed functions and application programming interfaces that provide a software platform for application
development. Refer to the Figure 3.1, which illustrates how the DAE, as a systems enabler, provides a software base. The API and services layer sit between the operating system and the application layer of the computer – PC, PS/2 or an IBM industrial computer.

The API consists of high-level functions that can be used by an application programmer. These API calls provide a consistent interface between the application program and the services layer (communications, data management, device management, etc.) of the DAE. They provide functional capability to easily develop and implement plant-floor applications for an industrial environment. As indicated in the Figure 3.1, the plant-floor applications sit at the application program level. These applications may use the API calls to avail the services provided by the DAE, or use the services provided by the operating system.

DAE makes use of blocks of information called control blocks. These control blocks are present at the Communications Systems level. Control blocks provide information on the location of files and plant-floor devices on the factory floor to the Communication System. These files and plant-floor devices constitute the system’s resources. The control blocks permit the system to hide the
Figure 3.1 DAE - Systems Enabler
information about the location of these system resources, thus leading to location independent code.

The Communications System has the high-level interfaces to the communications functions that are commonly required at a device-intensive, automated, manufacturing site. The above concept leads to network flexibility and data transparency in a plant-floor environment.

3.2 DAE and the Plant-Floor Environment

Plant-floor systems are moving towards distributed control architecture. This permits control to be maintained close to the location of the actual work. Thus work can be processed at the location where control is needed, either in the cell controller or in the area controller. DAE supports the above concept of distributed architecture by providing building blocks to configure a plant-wide network of integrated systems.

The information needs of devices on the plant-floor are met because of the communications flexibility provided by DAE. Through the use of these services, data can be located on any node of the network and accessed globally. Thus, it is possible to upload, download programs to devices from a controller located anywhere on the network.
Flexibility on plant-floor is required which will enable application and plant-floor devices to be moved to different locations without impacting the application programs. DAE makes this possible because the system resources are referred to by logical names rather than by physical locations, and this connection, between the logical and physical names, is provided through the control blocks.

3.3 DAE and VPI MARL

As mentioned previously, DAE is an application development tool that can be used to reduce custom programming and implementation. DAE was installed at the Manufacturing, Automation and Robotics Laboratory (MARL) of the Industrial and Systems Engineering Department of Virginia Tech to provide the required system wide integration. Also, a communication network based on DAE was setup, which would facilitate the management and communications between the devices at the work cell and area level control points.

The research needs at the Manufacturing, Automation and Robotics Laboratory are constantly changing. The use of DAE as a software platform would ease the expansion and modification of control strategies at the various work cells and area level control cells. It would reduce the user-learning curve, increase the programmer’s productivity, and
improve the application program's portability, thereby achieving integrated systems more rapidly.

The project development was done in the DAE environment. The intent of the research was to develop a device driver for the existing GE OITs. The system developed can be used with the large number of GE OITs located at the various work cells, thereby saving future programming efforts. The network communications component of DAE was used in driving the GE OIT from a remote node in the network. The developed system permits the user to perform operations on the GE OIT from any node on the network.

3.4 Basic Concepts of DAE

Three concepts are fundamental to a complete understanding of DAE and are explained below.

1. **API**: As already mentioned, an API provides an interface between the application programs and the services of the DAE. Functions that would normally be handled by the application are built into the interface, thus easing the task of implementing an integrated system.

An API is used by an application program to access the DAE services. DAE supports application development in C, PASCAL, and Assembler. The
application issues API requests to obtain system services, or to send messages and data to other applications and resources within the system. The API requests in turn make calls to functions or processes to perform these operations. Thus, the programmer is insulated from performing housekeeping tasks, because the APIs provide complex system management software.

APIs provide different types of control to the programmer, and are thus grouped into the type of control they provide. Some examples are program control, message control, data supply to application and devices, security processing, etc. APIs within a category provide services only related to their group. For example, APIs classified as program control requests enable programmers to control the execution of application programs like start, stop, purge, etc., whereas, APIs classified as message control requests are restricted to sending and receiving messages between programs. Apart from using the services provided by the APIs, an application programmer can also interact with the operating system directly and avail its services.
2. **Resource:** A resource is any identifiable hardware or software object, such as a manufacturing device, a system unit, a printer, an application program, etc. A manufacturing device can be a robot, or a programmable controller, a system unit can be an area level controller or a work cell controller and an application program can be a device driver or a program to be uploaded or downloaded to a machine. All DAE resources are identified by basic resource names which represent the logical name given to the resource either by the programmer or supplied by the Communications System. The resource names help to uniquely identify any resource within a network. Each type of resource has a corresponding control block that represents resources of that type. The control blocks contain information that is required by DAE to identify, locate, communicate with, and use the system resources.

All resources are defined to DAE through the Configuration Utility, where a control block for a resource is generated and a unique resource name assigned to it. The configuration utility lets the user access different types of system control data, edit, maintain and print reports about the data.
through a panel. Additionally, this option allows the user to access memory or disk-resident control blocks.

A control block for each defined resource is kept in a control block file. The control blocks in the control block file are used to build the resource directory when a node is initialized and a copy of the control block is brought into memory when the resource is started. A control block is used to store control data about the resource it represents, and also to maintain its status. They have fields where values for resource names, paths, initialization values, etc., can be entered, and these fields are defined by the user through the Configuration Utility.

3. **Network Communications:** Communications System permits efficient communications between plant-floor devices and the cell and area control points. The network communications control the transfer of data between nodes and are transparent to the application programs. The data transferred can be messages or requests.

The application programs may make use of the message control APIs to transfer messages anywhere in the
network. If the destination resource is located on the same node as the request originator is, then network communications component is not used. However, if the destination resource is on a different node, the network communications component takes over. The location of the destination resource is determined through the control block file, and the message or request is transferred to the desired node. Thus messages can be sent in a consistent manner regardless of their destination.

The network communications component eliminate the user program's need to know 1. the path to the destination resource; 2. physical location of the destination resource; 3. input and output between network device drivers and the local area networks; and 4. communications protocol between processing units.
CHAPTER 4. GE OPERATOR INTERFACE TERMINAL

The GE Operator Interface Terminal (GE OIT) is a 12" CRT terminal designed to be used in a harsh industrial environment. It is capable of displaying 25 rows and 80 columns of characters at one time and supports text and graphics display. It has a built-in keyboard that features a numeric keypad, a cursor control pad, and 15 function keys. Apart from the built-in keyboard, the GE OIT also supports an external connected keyboard.

4.1 GE Operator Interface Terminal Overview

The GE OIT supports most of the functions found in a plant-floor terminal, and is intended to be used with a wide variety of industrial machines that are capable of serial communications. These machines can include programmable controllers, CNC systems, robots, microcomputers and other control systems. However, the GE OIT derives its intelligence from the host system it is connected to, and in order to be used effectively, requires a device driver or control program. This control program usually resides in the host system and controls all data that is received and transmitted by the GE OIT. The GE OIT can be used as an operator interface in industrial applications requiring process monitoring, data entry, supervisory control, etc.
It has different options regarding data communication format and operating features and can replace many intelligent or non-intelligent terminals designed to follow either ANSI x3.64 or VT52 escape sequences.

The GE OIT can be classified as a fairly intelligent terminal, which means that in addition to transmitting and receiving data, it is able to respond to a wide variety of terminal commands. These commands are referred to as escape sequences. These escape sequences typically consist of the ASCII character ESCAPE followed by additional characters. These escape sequences identify a specific operation to be performed by the GE OIT like configuring, erase and edit function, or character attribute control. The different operations that can be performed by the GE OIT are outlined in the following sections. The GE OIT responds to the received escape sequences by performing the desired operation associated with it. The American National Standards Institute (ANSI) published a set of standards that recommends standard formats for commands to be used by vendors for new equipment. The GE OIT supports numerous commands in the ANSI format, thus making it compatible with existing software.

The GE OIT is intended to be used with a wide range of industrial machines which are capable of serial
communication at an RS-232-C or an RS-422 signal levels. The GE operator interface terminal has a Zilog Z80A processor and 32k bytes of user memory. There are a number of character attributes like blink, highlight which can be programmed. The GE OIT supports up to 161 graphics characters that can be clustered to form any number of displays. Among the preformed graphics are the quad size characters and process control symbols called Industrial Icons shown in Appendix C.

The OIT can be configured for various options regarding data communication formats and operating features. These options can be selected or changed by an operator using the set-up menu or through software control. These include Baud Rate selection, Parity, Full/Half Duplex, Escape Sequence type (ANSI or VT52), Communication type, etc. The OIT also contains a battery-backed clock and calendar which can be set or reset from the GE OIT keyboard or from a host computer over the communications link.

The GE OIT can operate in two modes, namely, the on-line mode and the local mode of operation. When in the local mode, the GE OIT responds to commands like setting character attributes, and other escape sequence commands from the OIT keyboard, and, commands transmitted over the serial interface from the host are ignored. The on-line mode can
be sub-divided into full duplex and half duplex operations. In the full duplex mode of operation, all keys entered from the OIT keyboard are transmitted to the host and no action is taken by the OIT unless the key codes are echoed back to the OIT by the host computer. In the half duplex mode of operation, all keys are transmitted to the host and simultaneously, action is taken by the OIT as if keys were received from the host.

Function key macros can be programmed to accomplish specific tasks and these macros can consist of display data or escape sequences. This option permits the user to perform operations (limited to 16 characters) by a single keystroke. The first 256 bytes of the GE OIT’s user memory is reserved for function key program storage. The function keys (F1-F15) perform different operations depending on which mode the GE OIT is in, either local or on-line. When in the local mode, they automatically select various character and line attributes. When in the on-line mode, the GE OIT requires that the function key be identified, prior to executing the program associated with the key. This identification of the function key is achieved by sending a special escape sequence to the GE OIT. This escape sequence for the ANSI standard is ESCAPE followed by the characters O(parameter) where the value of the parameter helps the GE OIT to differentiate between the function keys.
F1 to F15. Thus, the function key represented by the escape sequence will initiate the execution of a macro or program previously defined by the user in the on-line or local mode of operation.

User-defined screens can also be created and stored in the user memory of the OIT. These can be later displayed, added to or deleted by executing the appropriate command.

4.2 Operation and Functionality

As mentioned before, all operator interfaces recognize distinct character sequences for performing operations like cursor positioning, configuring, character attribute control, etc. These escape sequences can be grouped according to the control they provide. The following sections contain the major groups of software control supported by the GE OIT.

4.2.1 Special Keys

There exist a few keys which when depressed make the OIT respond in a special way. In the half duplex mode of operation, the keys are acted upon immediately by the GE OIT, whereas in the full duplex mode of operation, the GE OIT requires the keys to be retransmitted by the host computer. These keys are Break, Line feed, Back space, Return, Space, Delete, Repeat, Escape, and Reset.
4.2.2 Control Codes

The CONTROL key, when held down in conjunction with other keys, is used to send the 32 ASCII control codes. The OIT responds to only nine of these control characters. These are Bell, Back space, Tab, Line feed, Shift in, Shift out, Carriage return, Cancel and Escape.

4.2.3 Cursor Functions

Cursor movement is made possible by using either the cursor control keys or by transmitting special escape sequences. Cursor control keys (direction keys on the keyboard and the HOME key) will transmit their corresponding escape sequences when on-line. However, in the full duplex mode of operation, the cursor position on the GE OIT screen is unaffected, unless these escape sequences are echoed back to the OIT by the host. In the half duplex mode, immediate action is taken by the OIT to the cursor control keys.

Apart from the regular cursor control, which is provided by the cursor control keys, there exist special escape sequences which allow the user to address the cursor directly, save, restore and also report the current cursor position.
4.2.4 Erasing and Editing

The erase and edit feature allows the host computer to have control over the contents of the OIT screen and erase and edit information displayed on it. This feature allows the user to erase different sections of a line the cursor is currently positioned at, like, start of line to current cursor position, current cursor position to end of line or the entire line. Additionally, characters and lines can be inserted or deleted, and information on the screen cleared using special escape sequences.

4.2.5 Character Attributes

It is possible to enable various character attributes like reverse video, highlight video, dim video, shade video, blinking mode, underline mode, etc. for display purposes. It is also possible to enable or disable special graphic character sets. These graphic character sets enable the user to access and display special characters on the display screen of the GE OIT. These character sets are Character Set 1, Character Set 2 and Quad Size Character Set. However, prior to their use, these character sets need to enabled. There exist special escape sequences for enabling and disabling these character sets. These character attributes prove useful for creating effective screens for the operator interface.
There are special escape sequences associated with each of the character attributes and they can be used to accomplish the following: Highlight Video, Dim Video, Underscore, Blink, Reverse Video, Enter/Exit Supplemental Graphic Set 1, Enter/Exit Graphic Set 2, Shade Video, Double Wide Characters, Quad Size Characters, and Foreground/Background intensity.

4.2.6 Line Attributes

It is possible to set certain attributes on a line by line basis. There are special escape sequences associated with each of these line attributes and they can be used to accomplish the following: Single height, Double width single height, Single width double height, and Double width.

4.2.7 Clock and Date Functions

The GE OIT contains a battery-backed clock, which can be set or reset either locally or by the host computer. Numerous functions are possible with the battery backed clock and date feature of the OIT. It is possible to modify, display, and also transmit the time and date on the GE OIT to the host computer. There exist special escape sequences for clock and date functions, and the following operations can be performed: Set time, Set date, Request time, Request
date, Enable/disable Time display, and Enable/disable date display.

4.2.8 Split Screen Scroll

A scrolling region can be defined in terms of an upper boundary and a lower boundary. A special escape sequence accomplishes this.

4.2.9 File Storage Functions

Industrial applications require frequent use of screens or forms which are used by operators for monitoring processes or for data entry. These screens are typically stored in the host computer, and are transmitted to the plant-floor terminal every time the operator requires them. The GE 0IT has 32k bytes of user memory. This memory can be used for storing up to 255 screen files, the size of the files being limited only by the size of the memory. Instead of the host computer transmitting characters required for each screen display, screens can be displayed from the user memory by specifying the screen number assigned to each screen. These screen files can be designed and stored in the local mode or the on-line mode. Existing screen files can be later recalled, appended to, deleted or displayed. Special escape sequences are associated with each of the screen operations,
and these accomplish the following: Open screen file, Append screen file, Display screen file, and Save screen file.

4.2.10 Function Key Programming

The first 256 bytes of user memory are used for Function Key Program storage. It is possible to program the function keys to accomplish specific purposes, the length of the program being limited to 16 characters. The program can be displayable data, or any other valid escape sequence. Function key programming can be accomplished in the local or on-line mode of operation.
CHAPTER 5. DESIGN METHODOLOGY

5.1 Method of Approach

The general tasks required to accomplish the objectives of the system are listed below:

1. Gain familiarity with the GE OIT and identify all its programmable features,

2. Understand the DAE architecture, its control blocks and data structures that are required by it,

3. Software development.

In order to develop all the functional capabilities of the GE OIT, a thorough knowledge of the interface was required. The GE OIT’s features and operations were studied in detail and proper understanding of its functionality was gained.

Familiarity with the DAE architecture and its API calls were also gained. The data structures - Request Data Structure (RDS) and Resource Name Structure (RSN) and the control blocks play a very important role in the message passing mechanism employed by the DAE, and these were defined and used in developing the system.

The development and implementation of the system was done on two OS/2 based IBM industrial computers (7552), in the C programming language, using Microsoft C Version 6.00A. The
two system units were connected through a Token-Ring Network and configured to be the two nodes, Node1 and Node2 of the DAE. The Communications Manager of OS/2 provides API services, which enable application programs to communicate with other computers. The APPC REM provided by DAE, makes use of the API services of the Communications Manager for performing network communications. The communications network was configured using the OS/2 Communications Manager Systems Network Architecture Advanced Program-to-Program Communication (SNA APPC) protocol. A schematic description of the network configuration, consisting of the two industrial computers and the GE OIT is illustrated in Figure 5.1.

The software development was done in three phases as is explained below.

1. Development of library routines,

2. Development of the I/O control module for the controlling node, and

3. Development of a demonstration program.

In the first phase, the complete set of library routines were developed. The effort in this phase was directed towards creating a software tool, which will permit the future programmer to use the routines, without having an in-depth knowledge of the valid escape sequences recognized by
Figure 5.1 Schematic description of network configuration
GE OIT. By using the routines provided in the library, the programmer can utilize the GE OIT’s capabilities to its full extent. The library of routines can be used with the large number of GE OITs stationed at various work cell controllers in the Manufacturing Automation and Robotics Laboratory. The programmer can easily interface and use the GE OITs with minimal programming. The routines developed in this phase were grouped under the following categories:

1. Character Attribute Control
2. Line Attribute Control
3. Clock and Date Control
4. File Storage Control
5. Function Key Programming
6. Configuration Control
7. Cursor Control
8. Erase and Edit Control
9. Graphics Display

Each of the above categories has a set of routines associated with it. Each of the routines in the category achieved one or more of the programmable features under its control group. An example of a library routine is oit_cnf_model_1(option), which enables the programmer to configure the GE OIT to different options like select hardware/software handshake, select full/half duplex mode,
etc., by a simple function call and passing the right parameters. This function would permit the programmer to configure the GE OIT from any node of the network. The library routines, their valid parameters, and return codes are explained in detail in the User's Guide contained in Appendix F.

The second phase of the project was directed towards developing an I/O control module for the GE OIT. This module was located at Node 1 of the network. The Node 1 of the network served as a host controller for the GE OIT. It was connected by a serial communications link at RS-232-C signal level to the GE OIT. The two major tasks that were identified for the I/O control module are listed below:

1. Receive information from any node on the network and direct it to the GE OIT display screen, and

2. Receive messages from the GE OIT (input by an operator from the OIT keyboard) and direct it to a requesting program.

This module had an alias name defined for it, which is the "OIT_CONTROL". By defining an alias for the program, the GE OIT and its control module can be relocated to another node of the network when necessary, without modifying and recompiling the application program. As mentioned above, this module can receive information from any application.
program wanting to interface with the GE OIT. This is accomplished as follows. The application program wanting to display information or perform operations on the GE OIT, calls the required routine (developed in the first phase). The programmer needs to specify the destination resource as "OIT_CONTROL" in the program code and the library routine sends the required command or data to the I/O control module. The I/O control module receives the information and directs it to the GE OIT display screen. Also, an application program can request for information from an operator stationed at the GE OIT. The programmer calls the library routine responsible for receiving data from the GE OIT, and this routine internally makes the required DAE API calls which request the I/O control module to receive data from the GE OIT. The I/O control module receives data entered by the operator at the GE OIT keyboard, and directs it to the requesting application program. Thus, the GE OIT can send and receive messages from multiple sources on the network.

This module was defined to be an autoloading program. DAE provides an autoloading capability that permits a program to be loaded in the computer's memory on request. The autoloading program processes any pending requests and if no more requests are pending, releases resources and purges itself. By defining the program’s ACB to be autoloaded, the user at
Nodel was oblivious to its existence. This program "comes alive" only when required, eliminating the need for user initiation, and if no messages are currently pending in its ACB queue, purges itself from the computer's memory.

The I/O control module represents the only mechanism by which an application programmer can interface with the GE OIT. The programmer makes calls to the routines in the library, and these in turn communicate with the I/O control module. Additionally, the I/O control module was also responsible for setting the parameters like baud rate, parity, and number of data bits for the serial port on Nodel.

The third and final phase was concentrated towards the development of a demonstration program. In this program, the capabilities of the routines developed in the first phase were illustrated through a menu-driven program. The program demonstrated the following capabilities of the system:

1. Configuration of the GE OIT to different options from a remote node of the network,
2. Performing screen display operations on the GE OIT screen from a remote node of the network,
3. Receiving data entered by an operator at the GE OIT keyboard and routing it to a requesting application,
4. Performing graphics display on the GE OIT screen, and

5. Function key programming.
CHAPTER 6. SYSTEM DESCRIPTION

6.1 Introduction

In this chapter, the details of the system developed is described. First, the required hardware and software products are discussed, and this is followed by a section on the DAE programming environment. The section on DAE describes the control blocks, structures and API calls that were used in developing the system. Finally, the major modules of the system are presented.

6.2 Hardware Requirements

There are four hardware products required for the implementation of the project: System Unit, GE Operator Interface Terminal, IBM Token Ring Adapter, and the Network Communications.

Two system units (IBM Industrial Computer 7552) were installed and connected via a Token-Ring communications network. These units constituted the two nodes Node1 and Node2 of the DAE network.

A GE OIT was connected to one of the system units through a serial I/O interface at an RS-232-C signal level. All
communications between the GE OIT and the system unit was handled by this interface.

An IBM Token-Ring Network adapter II was installed on the two system units and connected through a network. The above adapter was required because the system unit used the network REM of DAE to access the network.

The network communication component of DAE transfers data between the different nodes on a network. The network communications component is the communications Resource Execution Module (REM). These are executable programs provided by DAE, and they provide an interface for a DAE node to the network driver. There must be a communications REM loaded at each node of the DAE network. Figure 6.1 illustrates the relationship between the communications REMs on the DAE nodes. The network device driver acts as an interface between the communications REMs and the network hardware.

The communications REM present on each node of the DAE node, send and receive Request Work Area (RWA) messages. The RWA is a device-independent information packet that is used by the internal processes of the system for communication between various application programs located on different nodes of the network. The application program is unaware of a request work area (RWA) message being delivered to a local
Figure 6.1 Node-to-Node Communications
or remote resource. Figure 6.2 illustrates the layout of a RWA. The communications REMs handle only the transmission portion of the RWA and ignore the data portion. The APPC REM of DAE was selected for communications, and its details are listed below.

<table>
<thead>
<tr>
<th>Communications</th>
<th>Communications Protocol</th>
<th>Network Adapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS2APR00</td>
<td>APPC</td>
<td>IBM Token-Ring Adapter II</td>
</tr>
</tbody>
</table>

6.3 Software Requirements

There are three software products required to implement the software system developed in this project: the OS/2 operating system, Language Support (MicroSoft C), and the Communications System/2 (DAE).

Communications Systems/2 (DAE) requires that the operating system OS/2 be installed on the system unit. The OS/2 version installed on the two system units was IBM Operating System/2 Extended Version 1.2. Also, the OS/2 Communications Manager was installed at each node of the network. The OS/2 Communications Manager provides API services that enable application programs to communicate with other computers on a network. The APPC services of the Communications Manager was used for communication. The DAE APPC Resource Execution Module (REM) which was selected, supports the above protocol of the Communications Manager.
P = Priority of message
F = Flags
C = Communications portion
S = Size of data portion of RWA, max size = 60,000 bytes

Figure 6.2 Layout of a Request Work Area
The software development was done using the Microsoft C Version 6.00A Optimizing Compiler. The DAE API calls support the above compiler.

The software development was done under the DAE environment. DAE was loaded at each node of the communications network. The DAE programming environment and its control blocks are described in the following section.

6.4 DAE Programming Environment

The Distributed Automation Edition environment (DAE environment) Version 2.0, was used for developing the system. The Communications System/2 (or DAE) comes with two network Resource Execution Modules (REMs) namely the APPC and NETBIOS. These REMs, let the Communication System/2 (CS/2) nodes communicate over a network. The APPC REM was selected for communication purposes. This REM lets the CS/2 nodes communicate over the IBM Token-Ring Network using the OS/2 Communications Manager SNA APPC protocol.

As mentioned earlier, all existing resources on the DAE network should be defined. Thus several control blocks were setup on each node of the DAE network. An Application Control Block was set up for the I/O Control Module - OIT_CTRL.CS2NODE on Node1 of the network, and another one for the demonstration program - DEMO.CS2NODE was defined on
Node2. An Alias Name Control Block (ANCE) was also set up for the program OIT_CTRL.CS2NODE. Apart from the above control blocks, several Communications Control Blocks (CCBs), Communication Access Control Blocks (CACBs) and Remote Control Blocks (RCB) were also defined on the DAE nodes. The details of all the control blocks on each node of the network are given in Table 6.3 and Table 6.4.

6.4.1 DAE API Calls

The Application Programming Interface (API) consists of all the function calls known as API requests that are available to an application programmer in the DAE environment. These API requests are grouped as follows.

1. Program control
2. Message control
3. Table and queue processing
4. Operator interface screen control
5. Direct and spooled printing
6. Security processing for restricted access
7. Tracing and logging
8. Alert handling and signaling
9. Data supply to applications and devices
10. Device data management support
### Control Blocks on Node1

<table>
<thead>
<tr>
<th>Control Block Type</th>
<th>Control Block Name</th>
<th>Location Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACB - Application Control Block</td>
<td>OIT_CTRL.CS2NODE</td>
<td>Current location of the executable file for the I/O Control Module.</td>
</tr>
<tr>
<td>CCB - Communications Control Block (Local)</td>
<td>NODE1.CS2AP000</td>
<td></td>
</tr>
<tr>
<td>CCB - Communications Control Block (Remote)</td>
<td>NODE2.CS2AP000</td>
<td></td>
</tr>
<tr>
<td>CACB - Communications Access Control Block</td>
<td>CS2AP000.CS2NODE</td>
<td></td>
</tr>
</tbody>
</table>
### Table 6.4 Control Blocks on Node2

<table>
<thead>
<tr>
<th>Control Blocks on Node2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACB - Application Control Block</td>
</tr>
<tr>
<td>Name of the control block: DEMO.CS2NODE</td>
</tr>
<tr>
<td>Location: Current location of the executable file for the demonstration program.</td>
</tr>
<tr>
<td>ANCB - Alias Name Control Block</td>
</tr>
<tr>
<td>Name of the control block: OIT_CONTROL</td>
</tr>
<tr>
<td>Physical Resource: OIT_CTRL.NODE1</td>
</tr>
<tr>
<td>RCB - Remote Control Block</td>
</tr>
<tr>
<td>Name of the control block: OIT_CTRL.NODE1</td>
</tr>
<tr>
<td>CCB - Communications Control Block (Local)</td>
</tr>
<tr>
<td>Name of the control block: NODE2.CS2AP000</td>
</tr>
<tr>
<td>CCB - Communications Control Block (Remote)</td>
</tr>
<tr>
<td>Name of the control block: NODE1.CS2AP000</td>
</tr>
<tr>
<td>CACB - Communications Access Control Block</td>
</tr>
<tr>
<td>Name of the control block: CS2AP000.CS2NODE</td>
</tr>
</tbody>
</table>
Only the program control requests and the message control requests were used in developing the system. Program control requests were used to start, initialize and stop the application programs. Message control requests were used to send and receive messages between the nodes of the network and also to start the autoload program (I/O control module) on the Node 1 of the network. The API calls that were used in developing the system are explained below.

**PROGRAM CONTROL REQUEST**

There are six program control requests that were used in developing the system:

**CS2_INIT(address of RDS)**

This request should be the first request issued by an application program before any other Communication System request. It indicates the successful initialization of the Communications System.

**CS2_INITIATE_RES(address of RDS)**

This request should be issued by an application program that requires the services of the Communications System. This request provides a mechanism for programs that are running under the operating system and are not started by the communications system to utilize its services.
CS2_WHO_AM_I(address of RDS, address of WHO_AM_I structure)
This request provides the program with its resource name. This name can be used by the program to specify the originator resource name in the RDS when it issues a request.

CS2_STOP_RES(address of RDS)
This request allows an application program to stop its own ACB, which inhibits further messages from getting queued in it.

CS2_PURGE_RES(address of RDS)
This request is issued by an application program to purge any requests currently residing on its control block queue. It also deletes the destination program’s control block from the control block pool, and sets the Purged status in the resource directory.

CS2_COND_PURGE(address of RDS)
This request is issued by an autoload program to delete its own ACB from the control block pool. The autoload program is required to check its ACB queue and ensure that no requests for services are awaiting it, before issuing this request.
MESSAGE CONTROL REQUESTS

There are four message control requests that were used in developing the system:

CS2_SEND(address of RDS, address of send structure)
This request is used by an application program to transmit messages to other resources, which could be located anywhere on the network. The message sent by the application program is queued in memory or the disk, depending on the disk-queuing threshold value in the destination program’s ACB or MCB. This request can also be used to start an autoloop program.

CS2_RECEIVE(address of the RDS, address of receive structure)
This request removes a message from the application program’s ACB queue or local MCB queue. The messages are retrieved from the appropriate queue, depending on the priority order.

CS2_SNDREC(address of RDS, address of send_receive structure)
This request combines the functions of the two requests CS2_SEND and CS2_RECEIVE in one single request. It allows a program to send a message and optionally receive a message from that resource.
CS2_REPLY(address of RDS, address of reply structure)

This request is used in conjunction with the CS2_SNDREC API call. It permits the application program to reply to the originator of CS2_SNDREC with an appropriate user-specified return code, and optionally, data.

6.5 System Modules

The two major modules that were developed are: 1. the library of routines, and 2. the I/O control module. This section describes the nine categories of the library functions that were developed. These nine categories are listed in Table 6.5, along with the associated routines. Each of these categories are responsible for a particular feature like configuration, character attribute control, cursor control, etc., and have a set of functions under them for setting or resetting attributes on the GE OIT. Additionally, the I/O control module is also discussed. As already mentioned, this module handles all the communications between the GE OIT and an application program on the network.

6.5.1 Library Routines

The various library routines that were developed are explained below. As explained earlier, they are grouped under the form of control each of them provide.
<table>
<thead>
<tr>
<th>Control Group</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Attribute Control:</td>
<td>oit_char_att_1,</td>
</tr>
<tr>
<td></td>
<td>oit_char_grph_1</td>
</tr>
<tr>
<td>Line Attribute Control:</td>
<td>oit_ln_att_1</td>
</tr>
<tr>
<td>Clock and Date Control:</td>
<td>oit_cal_set_1,</td>
</tr>
<tr>
<td></td>
<td>oit_cal_disp_1</td>
</tr>
<tr>
<td>File Storage Control:</td>
<td>oit_scr_open_1,</td>
</tr>
<tr>
<td></td>
<td>oit_scr_close_1,</td>
</tr>
<tr>
<td></td>
<td>oit_scr_append_1,</td>
</tr>
<tr>
<td></td>
<td>oit_scr_display_1,</td>
</tr>
<tr>
<td></td>
<td>oit_scr_fill_1</td>
</tr>
<tr>
<td>Function Key Programming:</td>
<td>oit_fnc_start_1,</td>
</tr>
<tr>
<td></td>
<td>oit_fnc_stop_1</td>
</tr>
<tr>
<td>Configuration Control:</td>
<td>oit_cfg_model1_1,</td>
</tr>
<tr>
<td></td>
<td>oit_cfg_mode2_1,</td>
</tr>
<tr>
<td></td>
<td>oit_cfg_mode3_1</td>
</tr>
<tr>
<td>Cursor Control:</td>
<td>oit_cur_mov_1,</td>
</tr>
<tr>
<td></td>
<td>oit_cur_posn_1,</td>
</tr>
<tr>
<td></td>
<td>oit_cur_svrs_1</td>
</tr>
<tr>
<td>Erase and Edit Control:</td>
<td>oit_edt_disp_1,</td>
</tr>
<tr>
<td></td>
<td>oit_edt_ln_1,</td>
</tr>
<tr>
<td></td>
<td>oit_edt_char_1</td>
</tr>
<tr>
<td>Graphics Display:</td>
<td>oit_grph_setatt_1</td>
</tr>
<tr>
<td></td>
<td>oit_grph_fillrg_1</td>
</tr>
<tr>
<td></td>
<td>oit_grph_drbox_1</td>
</tr>
<tr>
<td></td>
<td>oit_grph_bar_1</td>
</tr>
<tr>
<td></td>
<td>oit_grph_splscr_1</td>
</tr>
</tbody>
</table>
Character Attribute Control

Numerous character attributes can be set on the GE OIT. Once character attributes are set or reset, the change affects characters received by the GE OIT only after the particular function has been called. The existing characters on the display screen of the GE OIT are unaffected by the call. The functions in this group enable the application programmer to design effective displays on the GE OIT screen. Two functions have been defined which enable the programmer to set and reset character attributes on the GE OIT: oit_char_att_1 and oit_char_grph_1. These two functions handle all the programmable character attributes of the GE OIT. The set of attributes that can be programmed by the above functions are listed below.

1. Disable video
2. Highlight video
3. Dim video
4. Under score
5. Blinking
6. Reverse video
7. Shade video
8. Double wide characters
9. Select foreground intensity
10. Select background intensity
11. Enable/disable display of graphic characters set 1
12. Enable/disable display of graphic characters set 2
13. Enable/disable display of quad size characters

For the valid parameter settings of the two functions in this group, refer to the User's Guide contained in the Appendix F.

**Line Attribute Control**

It is possible to set or reset additional attributes on a line by line basis on the GE OIT. These line attributes may be used with character attributes explained earlier, except that double wide character attributes are superseded by double wide line attribute. These attributes remain set on the GE OIT until they are specifically cleared. Only one function has been defined for setting or resetting line attributes: oit_ln_att_1. This function handles all the programmable line attributes of the GE OIT. The set of attributes that can be programmed by the above function is listed below.

1. Disable line attributes
2. Single height, double width
3. Single height, single width
4. Double height, double width
For the valid parameter settings of the function, refer to the User's Guide contained in the Appendix F.

Clock and Date Control

The GE OIT contains a battery-backed clock and calendar. This clock is always running even if the power is turned off. The GE OIT clock can be set or reset during terminal configuration, or by using special escape sequences sent from the host. When the clock and date display is enabled, the time and date are displayed on the status line, which is the lower right hand corner of the GE OIT screen. Two functions have been defined which enable the programmer to set and display the battery-backed clock and calendar of the GE OIT: oit_cal_set_1 and oit_cal_disp_1. The set of attributes that can be programmed by the above functions are listed below.

1. Set time, date
2. Display time, date

For the valid parameter settings of the two functions in this group, refer to the User's Guide contained in the Appendix F.
File Storage Control

The GE OIT has 32k bytes of user memory and this memory may be used to store user defined screens and function key programs. The memory itself is organized as files numbering from 0 through 254, and these files may be accessed in any order. Thus the memory can be used for storing up to 255 files, the size of the files being limited only by memory.

Six functions have been defined under this group: oit_scr_open_1, oit_scr_close_1, oit_scr_append_1, oit_scr_display_1, oit_scr_fill_1, and oit_scr_reinit_1. These functions handle all the programmable screen display features of the GE OIT and these are listed below.

1. Open screen file
2. Append screen file
3. Display screen file
4. Save screen file
5. Reinitialize screen file
6. Data fill operation

For the valid parameter values of the six functions in this group, refer to the User's Guide contained in the Appendix F.
Function Key Programming

The first 256 bytes of the GE OIT's user memory is used for function key program storage. It is possible to program the function keys to accomplish specific purposes, the length of the function key program being limited to 16 characters. The program can be displayable data, or any other valid escape sequence. The user can enter up to 16 characters and identify them to be associated with a particular function key. The predefined function key program can be executed in two ways:

1. The host can transmit the escape sequence which identifies the key to the GE OIT and the user defined program associated with the function key will be executed, or

2. The operator sitting at the GE OIT can press the function key when the GE OIT is in the on-line, half duplex mode of operation and the user defined program associated with the function key will be executed. However, if the GE OIT is configured for the on-line, full-duplex mode of operation, then the function key should be echoed by the host computer back to the GE OIT in order for the function key program to be executed.
Two functions have been defined which enable the programmer to perform function key programming from the host computer: oit_fnc_start_1 and oit_fnc_stop_1. These functions enable an application programmer to define and store function key programs in the GE OIT's user memory.

Configuration Control

The GE OIT has several different options regarding data communication format and operating features. Several of these operating modes namely the baud rate, parity, communication type - point to point or multidrop, status line setting, and local/on-line mode of operation can only be set in the local mode of operation of the GE OIT. Apart from the above modes, the remaining of the configuration modes namely, cursor type - block or blinking, auto line feed option, full or half duplex mode, VT52/ANSI escape sequence, auto wrap option, enable/disable keyboard, and enable/disable display are software programmable. Three functions have been defined which enable the programmer to configure the GE OIT to these different options: oit_cfg_mode1_1, oit_cfg_mode2_1, and oit_cfg_mode3_1. These three functions handle all the programmable configurations that the GE OIT can be set to, and these configuration options are listed below.

1. Cursor type - block, or underscore
2. Enable/disable cursor display
3. Automatic linefeed and carriage return
4. Full/half duplex mode
5. Enable/disable blinking cursor
6. Enable/disable cursor keys
7. Escape sequence type - ansi, or vt52
8. Enable/disable wrap
9. Enable/disable keyboard
10. Enter/exit insert mode

The valid parameter values for the three functions in this group are in the User’s Guide contained in the Appendix F.

**Cursor Control**

The GE OIT cursor movement is software controllable. Thus the application programmer can position the cursor anywhere on the screen, save and restore the cursor position, and also report the current cursor position to the host computer. Three functions have been defined which enable the programmer to control the cursor movement on the GE OIT screen: oit_cur_mov_1, oit_cur_posn_1, and oit_cur_svrs_1. These functions handle all the cursor control features that are programmable on the GE OIT and these are listed below.

1. Cursor home
2. Cursor up
3. Cursor down
4. Cursor forward
5. Cursor backward
6. Absolute cursor positioning
7. Save and restore cursor position

The parameter associated with the three functions in this group and their values are explained in the User's Guide contained in the Appendix F.

Erase and Edit Control

Editing of the GE OIT screen is possible using software control. The programmer can have control over the contents of the display and perform editing from his or her host computer. The application programmer can clear screen, insert and delete lines, insert and delete characters, and erase lines. Three functions have been defined which enable the programmer to perform erasing and editing functions from the host computer: oit_edt_disp_1, oit_edt_in_1, and oit_edt_char_1. These functions handle all the erase and edit features that are programmable on the GE OIT. These features are listed below.

1. Clear screen
2. Clear line
3. Delete line
4. Delete character

The parameter associated with the three functions in this group and their values are explained in the User's Guide contained in the Appendix F.

Graphics Display

Five functions have been included in this group, and these functions are intended to make screen programming easier for the user. The applications programmer can draw bar graphs, change attributes in a particular region, draw a box, define a scroll region or a solid region easily by using these functions. These functions have the character sequence 'grph' as the secondary element in their function name to distinguish them as functions belonging to the Graphics Display group. The list of options possible with the five functions in this category are listed below.

1. Set character attributes in a region
2. Fill a region with a character
3. Draw a box outline for a region
4. Draw horizontal and vertical bar graphs
5. Define a split scroll region
6.5.2 I/O Control Module

The I/O control module was defined to be an autoloading program. It came into existence only when it had pending requests or messages on its ACB queue. This program sits on a node which is connected to the GE OIT terminal through a serial I/O interface at RS-232-C signal level.

This program checked for two types of CS2 requests on its ACB queue - CS2_SEND() and CS2_SNDREC(). The request CS2_SEND was used by the library routines to send messages to the GE OIT through the I/O control module, and the request CS2_SNDREC was used by the library routines to receive data from the GE OIT through the I/O control module. When either of the above requests was received by the I/O control module, the module "came alive" and performed the required operations.

All the initializations required for the COM1 port on Model of the network are handled by the function init_port(). The required parameters for serial port initialization like baud rate, parity, and number of data bits are set using the functions DosDevIOCtl(), DosWrite(), DosRead(), and DosBufReset() provided in the OS/2 library. The I/O control module functions by receiving requests from its ACB Queue. It processes the request depending on whether data needs to be output to the GE OIT or input from the GE OIT. If data
needs to be output, the function `drv_out_data()` is called. This function accesses its queue for the data, and outputs the required string of data (control data or display data) on the GE OIT. If data needs to be input, then the function `drv_in_data()` is called. This function takes input data from the GE OIT, which is entered by the operator, and stores it in its queue, to be retrieved later by the originator of the request.

The function `getmsg()` receives pending requests and messages from its ACB queue. It issues the API call `CS2_RECEIVE` and receives the awaiting messages in its receive structure. The order of the received messages is based upon the priority set by the application program issuing the request. The function `check_pend_req()` is used check the ACB queue of the I/O control module for pending requests, and if no requests are pending then the program is purged from the control block pool. The API call `CS2_COND_PURGE` is issued by this function to determine if there are any pending requests in its ACB queue. This API request purges the program’s ACB from the control block pool, only if there are no pending requests. However, if any requests are pending, then the ACB is not purged. The major functions of this module are listed in Table 6.6.
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>drv_in_data()</td>
<td>Receives data input by an operator at the GE OIT</td>
</tr>
<tr>
<td>drv_out_data()</td>
<td>Sends data to the GE OIT screen</td>
</tr>
<tr>
<td>check_pend_req()</td>
<td>Checks the ACB queue for pending requests</td>
</tr>
<tr>
<td>get_msg()</td>
<td>Receives the requests from the ACB queue</td>
</tr>
<tr>
<td>init_port()</td>
<td>Performs the required initialization for the serial port</td>
</tr>
</tbody>
</table>
The demonstration program provides the future user of the system with sample code and shows the correct usage of the functions provided in the library. It also shows how the future user should initialize the required structures for the DAE API calls. Additionally, it demonstrates the capability of the library functions. The demonstration program was loaded and tested from Node2 of the network.

The demonstration program is a menu driven program, and depending on the user input, either displays a screen, or performs an OIT operation. The menu has 2 levels. The highest level is the main menu where the user selects the type of operation to be performed. Depending on the operation selected by the user, a suitable sub-menu is displayed. The sub-menu displays a list of valid operations which can be performed at the OIT. The user enters the number associated with the operation, and this causes a function to be executed. For example, if the user selects to configure the GE OIT, then a sub-menu with a list of options is displayed. The user can select to configure the OIT for the following modes: Half Duplex Mode, Full Duplex Mode, ANSI Escape Sequence Mode, VT52 Escape Sequence Mode, Hardware Handshake Mode, Software Handshake Mode or exit back to the main menu. When any of the above options are
chosen, a call is made to the appropriate function and the GE OIT operation is performed.

The main menu screen is illustrated in Figure 6.7. The user is presented with a menu which lists the operations that have been selected for demonstration purposes. These options serve to illustrate some of the system capabilities that were developed. The different options are: Configure the OIT, Perform Screen Operations, Graphics Display, or Function Key Operations. Selecting any of the above options, brings up a suitable sub-menu at the lower level. The user can choose to exit the program by choosing option 5.

When the user selects to configure the OIT, a sub-menu is displayed. This sub-menu is illustrated in Figure 6.8. The following options are displayed: Half Duplex Mode, Full Duplex Mode, ANSI Escape Sequence Mode, VT52 Escape Sequence Mode, Software Handshake Mode and Hardware Handshake Mode. For a complete set of configuration options, please refer to the User' Guide contained in the Appendix F. The use of the following two functions oit_cfg_mode1_1() and oit_cfg_mode2_1, with different parameters is demonstrated. The user can exit this sub-menu by choosing option 7, and this will take him or her back to the main menu.
MAIN MENU

1. Configuring OIT
2. Screen Operations
3. Graphics Display
4. Function Key Operations
5. Exit

Enter Option:

Figure 6.7 Main Menu Screen
CONFIGURE THE OIT

1. Half Duplex Mode
2. Full Duplex Mode
3. ANSI Escape Sequence Mode
4. VT52 Escape Sequence Mode
5. Software Handshake Mode
6. Hardware Handshake Mode

7. Exit to Main Menu

Enter Option:

Figure 6.8 Configuration Menu Screen
If the user chooses to perform screen operations from the main menu, a screen operations sub-menu is displayed. This sub-menu is illustrated in Figure 6.9. The user is presented with the following options: Display Sample Screen, and, Receive Sample Data from the OIT. For a complete set of screen operations that can be performed please refer to the User's Guide contained in the Appendix F. If the user selects the first option, then the function oit_scr_display_1() is called, and, this function displays a sample screen which had been programmed previously. This sample screen had been created using the function pgm_scr1(). This option illustrates how data can be displayed on the GE OIT screen from Node2 of the communications network. If the user selects the second option - Receive Sample Data, then the function oit_scr_display_1() is again called, and, this function displays a second sample screen which had been previously programmed. This screen had been created using the function pgm_scr3(). This option demonstrates how the data input by a user at the GE OIT keyboard can be received at Node2 of the network. After the sample screen is displayed on the GE OIT display screen, the program waits for operator input by calling the function rcv_data(). This function calls the library function oit_send_rcv_1() with the first parameter being RECEIVE, and receives the data entered by the operator.
SCREEN OPERATIONS

1. Display Sample Screen
2. Receive Sample Data from OIT
3. Exit to Main Menu

Enter Option:

Figure 6.9 Screen Operations Menu Screen
at the GE OIT keyboard and displays it on Node2. The user can exit this sub-menu by choosing option 3 and this will take him or her back to the main menu. Figure 6.10 and Figure 6.11 lists all the library functions that were used in this option.

The third option demonstrates the purpose of the graphic and character attribute functions provided in the library, and also shows how they can be used to program effective display on the GE OIT screen. This sub-menu is illustrated in Figure 6.12. Here the user is presented with the following options: Horizontal Bar Graph Display, Vertical Bar Graph Display and Flexible Manufacturing Cell Layout. Figure 6.13 and Figure 6.14 lists all the library functions that were used in this option.

The fourth option demonstrates the use of the function key operations on the GE OIT. This sub-menu is illustrated in Figure 6.15. Here the user is presented with the following two options: Program Function Key, and Execute Function Key Program. The use of the four library functions oit_fnc_start_1(), oit_fnc_stop_1(), oit_send_rcv_1() and oit_edt_disp_1() are demonstrated in this option. The user can choose to exit back to the main menu by choosing option 3.
Table 6.10 Functions Used for Programming Screen 1

<table>
<thead>
<tr>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oit_scr_open_1(1)</td>
</tr>
<tr>
<td>oit_edt_disp_1(CLR_SCR)</td>
</tr>
<tr>
<td>oit_char_att_1(DISABLE_VID)</td>
</tr>
<tr>
<td>oit_cur_posn_1()</td>
</tr>
<tr>
<td>oit_grph_drbox_1()</td>
</tr>
<tr>
<td>oit_char_att_1(HGLT_VID)</td>
</tr>
<tr>
<td>oit_char_att_1(REV_VID)</td>
</tr>
<tr>
<td>oit_send_rcv_1()</td>
</tr>
<tr>
<td>oit_ln_att_1(CLEAR_LNATT)</td>
</tr>
<tr>
<td>oit_scr_close_1()</td>
</tr>
</tbody>
</table>
Table 6.11 Functions Used in Receive Sample Data Option

oit_scr_display_1()
oit_cfg_model_1(DIS_CUR_KEY,ON)
oit_cur_posn_1()
oit_send_rcv_1()
oit_scr_open_1(1)
oit_edt_disp_1(CLR_SCR)
oit_char_att_1(DISABLE_VID)
oit_cur_posn_1()
oit_grph_drbox_1()
oit_char_att_1(HGLT_VID)
oit_char_att_1(REV_VID)
oit_ln_att_1(LN_SNHT_DBWD)
oit_ln_att_1(CLEAR_LNATT)
oit_scr_close_1()
oit_cfg_model_1(DIS_CUR_KEY,OFF)
GRAPHICS DISPLAY

1. Horizontal Bar Graph Display
2. Vertical Bar Graph Display
3. Flexible Manf. Cell Layout

4. Exit to Main Menu

Enter Option:

Figure 6.12 Graphics Display Menu Screen
Table 6.13 Functions Used in Bar Graph Display

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>oit_cur_posn_1()</td>
</tr>
<tr>
<td>oit_send_rcv_1()</td>
</tr>
<tr>
<td>oit_scr_open_1(1)</td>
</tr>
<tr>
<td>oit_edt_disp_1(CLR_SCR)</td>
</tr>
<tr>
<td>oit_char_att_1(DISABLE_VID)</td>
</tr>
<tr>
<td>oit_grph_bar_1(HORIZONTAL_BAR)</td>
</tr>
<tr>
<td>oit_grph_bar_1(VERTICAL_BAR)</td>
</tr>
<tr>
<td>oit_grph_drobox_1()</td>
</tr>
<tr>
<td>oit_scr_display_1()</td>
</tr>
<tr>
<td>oit_scr_close_1()</td>
</tr>
</tbody>
</table>
Table 6.14 Functions Used in Flexible Manufacturing Cell Layout Option

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>oit_cur_posn_1()</td>
</tr>
<tr>
<td>oit_send_rcv_1()</td>
</tr>
<tr>
<td>oit_scr_open_1(1)</td>
</tr>
<tr>
<td>oit_edt_disp_1(CLR_SCR)</td>
</tr>
<tr>
<td>oit_char_att_1(DISABLE_vid)</td>
</tr>
<tr>
<td>oit_grph_drbox_1()</td>
</tr>
<tr>
<td>oit_char_att_1(GRAPHIC_SET1,ON)</td>
</tr>
<tr>
<td>oit_char_att_1(GRAPHIC_SET1,OFF)</td>
</tr>
<tr>
<td>oit_char_att_1(GRAPHIC_SET2,ON)</td>
</tr>
<tr>
<td>oit_char_att_1(GRAPHIC_SET2,OFF)</td>
</tr>
<tr>
<td>oit_scr_display_1()</td>
</tr>
<tr>
<td>oit_scr_close_1()</td>
</tr>
</tbody>
</table>
FUNCTION KEY OPERATIONS

1. Program a Function Key
2. Execute a Function Key Pgm
3. Exit to Main Menu

Enter Option:

Figure 6.15 Function Key Operations Menu Screen
CHAPTER 7. CONCLUSIONS AND RECOMMENDATIONS

The control software for the GE OIT in the DAE environment was successfully developed in this project. The library of functions created for the GE OIT will aid the programmer to easily interface and communicate with the GE OIT. By using the library functions, the future user is insulated from the lower level functioning of the GE OIT, and need only be concerned about the operations of these functions. The use of the DAE environment permitted communication between the GE OIT located at a work cell controller and a node functioning as the system level controller. Thus the GE OIT could send and receive information from multiple sources on the network.

Major effort of this research was directed towards creating a library of functions which controlled the operation of the GE OIT. This included functions to configure the terminal, perform screen operations, control character and line attributes, perform function key programming, control the cursor position, perform graphics display, perform erase and edit functions, and control clock and time display.

An I/O control module was also developed. This module represented the only interface between the GE OIT and the library routines. Also, this module made it possible for
the GE OIT to send and receive data from multiple sources on the network. It was defined by an alias to permit future relocation of the GE OIT to a different node of the network, without requiring any modifications to the program. Finally, a demonstration program was developed to provide the programmer with sample code for the correct use of the library functions.

The present system only receives data from the GE OIT and does not perform any processing or validation of this received data. The current system permits the programmer to design new screens, append to existing screens and erase existing screens. These screens are stored in the user memory of the GE OIT. The above screens can be recalled and displayed on the GE OIT. The operator stationed at the GE OIT can enter data at the specified fields of the screen, the location and data type of the fields entirely dependent on the screen being displayed. Thus the data received from the GE OIT is very application specific. Future work should be directed towards providing an interface for the screen operations. This interface would make use of the library functions to perform screen operations like defining new screens, appending to existing screens, etc., and in addition provide further capabilities like defining data types, validating received data, etc. The interface should develop special data structures for the programmed screens,
so that these structures can be used for storing the screen
details like number of fields, data type of fields, etc..

Screen and function key programming is a time consuming
operation on the GE OIT. Thus, functions which were
developed for performing these operations should be
identified and stored in a separate program. In case the
user memory of the GE OIT should get corrupted, the program
can be used to reinitialize the user memory to its previous
state.
LIST OF REFERENCES


APPENDIX A. GRAPHICS CHARACTER SET

STANDARD ASCII CHARACTERS USED

STANDARD ASCII CHARACTERS USED

STANDARD ASCII CHARACTERS USED

STANDARD ASCII CHARACTERS USED

STANDARD ASCII CHARACTERS USED

SUPPLEMENTAL CHARACTER SET
APPENDIX C. QUAD SIZE CHARACTER SET

INCLUDED IN QUAD SIZE CHARACTER SET ARE:
- CAPITALS: A TO Z
- NUMERALS: 0 TO 9
- SYMBOLS: . , + - / ? ! SPACE
APPENDIX D. RDS FIELDS

Request Data Structure fields:

<table>
<thead>
<tr>
<th>orig</th>
<th>dest</th>
<th>ack</th>
<th>timeout</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnretcd</td>
<td>osretcd</td>
<td>msglen</td>
<td>seqnum</td>
<td>trace template</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig</td>
<td>The resource name structure contains the resource name of the originator of this request.</td>
</tr>
<tr>
<td>dest</td>
<td>The resource name structure contains the resource name of the destination of this request.</td>
</tr>
<tr>
<td>ack</td>
<td>User can designate if acknowledgement for a receipt is required or not.</td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout in millisecs for which the system will wait for an acknowledgement.</td>
</tr>
<tr>
<td>priority</td>
<td>Requests are queued according to their priority number. Priority can range from 10 thru 127, with 10 being the highest priority.</td>
</tr>
<tr>
<td>cnretcd</td>
<td>The CS/2 return code specifies the results of a request.</td>
</tr>
<tr>
<td>osretcd</td>
<td>Operating System return code field.</td>
</tr>
<tr>
<td>cmdncod</td>
<td>Indicates the command code of the request that has been received.</td>
</tr>
<tr>
<td>msglen</td>
<td>Specifies the size of the message that is received.</td>
</tr>
<tr>
<td>seqnum</td>
<td>Sequence number assigned by the system and passed in the RDS.</td>
</tr>
<tr>
<td>trans</td>
<td>Reserved</td>
</tr>
<tr>
<td>rdstype</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
id  Reserved
trace Lets the user trace the transmittal of a request.
template Template options structure.
encode Reserved
APPENDIX F. USER'S GUIDE
# TABLE OF CONTENTS

1.0 CHAPTER 1. INTRODUCTION ..........................1
1.1 General Description ..............................1

2.0 CHAPTER 2. FUNCTIONAL CAPABILITIES ..........3
2.1 Overview ........................................3
2.2 Grouping of Functions ............................3

3.0 CHAPTER 3. OIT FUNCTION LIBRARY ..............7
3.1 Overview ........................................7
3.2 Character Attribute Control ......................7
3.3 Line Attribute Control ...........................15
3.4 Clock and Date Control ...........................18
3.5 File Storage Control ..............................21
3.6 Function Key Programming .........................25
3.7 Configuration Control ............................27
3.8 Cursor Control ..................................33
3.9 Erase and Edit Control ..........................36
3.10 Graphics Display ..................................39
3.11 Misc Function ..................................42

4.0 CHAPTER 4. I/O CONTROL MODULE ...............45
4.1 Specifications ....................................45
5.0 CHAPTER 5. DEMONSTRATION PROGRAM ....................48
5.1 Overview .................................................48
5.2 Required Control Blocks .................................48
5.3 Processing ................................................52

6.0 CHAPTER 6. REQUIRED HARDWARE AND SOFTWARE PRODUCTS.65
6.1 Hardware Requirements .................................65
6.2 Software Requirements .................................66
6.3 DAE Programming Environment .......................67
LIST OF ILLUSTRATIONS

5.1 Control Blocks on Node1  ..............................49
5.2 Control Blocks on Node2  ..............................51
5.3 Main Menu Screen  .................................54
5.4 Configuration Menu Screen  ..........................55
5.5 Screen Operations Menu Screen  .....................56
5.6 Functions Used for Programming Screen 1  ..........58
5.7 Functions Used in Receiving Sample Data Option  ...59
5.8 Graphics Display Menu Screen  ......................60
5.9 Functions Used in Bar Graph Display  ...............62
5.10 Functions Used in Fl. Manf. Cell Layout Option  ...63
5.11 Function Key Operations Menu Screen  ...............64
6.1 Control Blocks on Node1  ..............................69
6.2 Control Blocks on Node2  ..............................70
CHAPTER 1. INTRODUCTION

1.1 General Description

The system provides all functional capabilities of the GE Operator Interface Terminal in the form of a library of routines. This library of routines provides a software platform which is available to all future programmers interfacing with the GE OIT. By making calls to the routines, and passing the right parameters, an application programmer will be able to interface the GE OIT to his or her system without worrying about interface details like opening ports, closing ports, baud rate, signal level etc. The routines were developed using the Microsoft C Version 6.00A Optimizing Compiler and they made extensive use of the Application Program Interface (API) calls of the DAE environment. The API calls were used for starting and stopping a program, and for sending and receiving messages between programs. By developing the code in the DAE environment, it was possible for an application program sitting on any node of the communications network to communicate with the GE OIT.

The GE OIT library routines provided are available in the library file oit_lib.lib. The library file should be linked with the source program file. The file oit_cnst.h contains
all the constant declarations that are required by the library routines and this should also be included in the source program file.

A demonstration program was developed to demonstrate the functional capabilities of the GE OIT and the communications flexibility provided by the system. The demonstration program shows the capability to display data, receive data and configure the GE OIT from any node of the network. It provides sample code for a future programmer attempting to use the GE OIT in the DAE environment. It shows how the library routines provided by the system can be used to do the following:

1) Configure the GE OIT to different options,
2) Perform Screen Operations,
3) Perform Graphics Display,
4) Perform Function Key Programming.

The demonstration program also serves as an example of how the DAE API calls can be used for developing a system.
CHAPTER 2. FUNCTIONAL CAPABILITIES OF THE SYSTEM

2.1 overview

The use of the DAE environment makes it possible for an application program sitting on a local or remote node of the CS/2 network to display data on the GE OIT or receive data from the GE OIT. Any application program sitting on any node of the CS/2 network can send or receive messages from the GE OIT by linking with the library developed, making the appropriate call and identifying the required parameters.

A programmer attempting to interface with the GE OIT can make calls to the functions provided in the library. The programmer can link his or her application to the appropriate library and make simple function calls to the routines provided and can thus receive data from the GE OIT or display data on the GE OIT without bothering about interface details.

2.2 Grouping of functions

Numerous functions will be made available to a programmer in the library provided. Each of these functions provide the programmer with the capability to set a specific attribute or feature on the GE OIT. These attributes or features are software programmable on the GE OIT. Some examples of the
programmable attributes are: character attribute, line attribute, cursor positioning, function key programming, file storage, etc. The functions are grouped according to the functional capability provided by them. These functions will make programming simpler for an application programmer wanting to display or receive data from the GE OIT. The following list shows the general groupings of the functions that provide different types of control.

1) Character Attribute Control
2) Line Attribute Control
3) Clock and Date Control
4) File Storage Control
5) Function Key Programming
6) Configuration Control
7) Cursor Control
8) Erase and Edit Control
9) Graphics Display

These are explained in detail below.

Character Attribute Control - Numerous character attributes can be set on the GE OIT using the functions provided under this group.

Line Attribute Control - Characters can have numerous attributes and these attributes
of characters can be set on a line by line basis. Functions related to line attributes are provided here.

Clock and Date Control - Functions related to clock and date are provided under this group.

File Storage Control - Functions related to file storage and display are provided under this group.

Function Key Programming - Functions related to function key programming are provided under this group.

Configuration Control - Helps the operator configure the GE OIT according to his or her requirements.

Cursor Control - Functions related to cursor positioning are provided here.

Erase and Edit Control - Provides erase and edit functions.

Graphics Display - Some additional functions have been defined here to make graphics display easier for the user.
Each of the above groups has a set of functions associated with it, and their functions details, parameters and return codes are outlined in detail in Chapter 4. Each of the above groups of functions are defined in a certain way so that they can be easily distinguished from each other. All functions are defined with a primary character sequence 'oit', and are followed by a 2, 3 or 4 letter character sequence denoting the group the function belongs to. For example, all functions defined under character attribute control have a 'char' character sequence as the secondary element in the function name and all functions provided in the configuration group have a 'cfg' character sequence as the secondary element in the function name.
CHAPTER 3. OIT FUNCTION LIBRARY

3.1 Overview

The library functions that were developed are explained in detail here. These functions are grouped according to their functionality as seen in Chapter 3. The basic capability of each, and parameters details (if any) are also explained here. All the functions in the library have the character sequence 'oit' as the primary element of the function name. This is followed by a three or four character long secondary element, which associates each function with its control group.

3.2 Character Attribute Control

Numerous character attributes can be set on the GE OIT. Once character attributes are set or reset, the change affects characters received by the GE OIT only after the function has been called. The existing characters on the display screen of the GE OIT are unaffected. Two functions have been defined which enable the programmer to set and reset character attributes on the GE OIT. These functions have the character sequence 'char' as a secondary element in their function name to distinguish them as Character Attribute Control functions. Thus the function name
'oit_char_aaa_1' indicates that it is a function associated with character attribute control on the GE OIT.

void oit_char_att_1(int option)
This function lets a programmer set and reset certain video attributes on the GE OIT. By making a call to this function, character attributes like highlight video, dim video, reverse video, underscore, blink, shade, and double wide characters attributes can be set or reset on the GE OIT. The parameter option specifies the character attribute to be set or reset. The function does not affect graphic or quad size character attributes (they have a separate function for enabling and disabling them). In order to set more than one character attribute, the function has to be called more than once. For example if the user wants the characters on the display screen of the GE OIT to blink and also be displayed in reverse video, then the function has to be called twice, once to set the characters to reverse video and the second time to enable the blinking mode. It is recommended that this be the first function called by an application programmer (with option set to DISABLE_VID) so that any previously set character attributes can be disabled. Valid options for this function and their explanations are given below.
**option: DISABLE_VID**

This option disables character attributes like highlight video, dim video, underscore video, blink mode, reverse video and shade video. If these attributes have been set previously on the GE OIT, they can be disabled by calling this function with option set to this DISABLE_VID.

**option: HGLT_VID**

This option displays characters on the GE OIT screen with higher than normal intensity. Characters received by the GE OIT after this option has been set, will be displayed with a higher than normal intensity. This mode can be disabled by calling function oit_char_att_1 with option set to DISABLE_VID.

**option: DIM_VID**

This option displays characters on the GE OIT screen with lower than normal intensity. Characters received by the OIT after this option has been set will be displayed with a lower than normal intensity. This mode can be disabled by calling function oit_char_att_1 with option set to DISABLE_VID.

**option: UNDER_SCR**

This option, when set causes underlined characters to be displayed on the GE OIT screen. Characters received by the GE OIT after this option has been set will be underlined and
displayed. This mode can be disabled by calling function oit_char_att_1 with option set to DISABLE_VID.

option: BLINK_ON
This function causes characters received by the GE OIT to blink on the screen. Characters received by the GE OIT after this option has been set will start to blink on the GE OIT display screen. This mode can be disabled by calling function oit_char_att_1 with option set to DISABLE_VID.

option: REV_VID
This option causes characters to be displayed on the GE OIT in reverse video. The characters displayed on the GE OIT screen appear as a dark character on a light background. The background intensity will be the same as previously used for foreground on the terminal (dim, normal, or highlight). Graphic characters can also be displayed in reverse video. This mode can be disabled by calling function oit_char_att_1 with option set to DISABLE_VID.

option: SHD_VID
This option, when set causes characters to be displayed on the GE OIT screen as a highlighted foreground with a dim intensity background. Characters received by the GE OIT after this option has been set will be displayed as such. This option can be disabled by calling function oit_char_att_1 with parameter set to DISABLE_VID.
**option: DBL_WIDE_ON**

Double wide characters can be displayed on the GE OIT screen on a character by character basis. This option should be used if double wide characters need to be displayed on the GE OIT. This character attribute can be disabled by the parameter `DBL_WIDE_OFF`.

**option: DBL_WIDE_OFF**

Double wide character attribute mode, if previously enabled on the GE OIT, can be disabled by this option. This is the only method by which the `DBL_WIDE_ON` option can be disabled.

**option: FG_UNDLM_DIM**

This option allows the user to dim the foreground intensity and display underlined characters. All characters received by the GE OIT screen after this option has been set will be underlined and displayed with a lower than normal intensity. This option can be disabled by calling function `oit_char_att_1` with parameter set to `DISABLE_VID`.

**option: FG_UNDLM_HIGHLT**

This option allows the user to increase the foreground intensity. All characters received by the GE OIT screen after this option has been set will be underlined and displayed with a higher than normal intensity. This option
can be disabled by calling function oit_char_att_1 with parameter set to DISABLE_VID.

**option: BK_HIGHINT**
This option allows the user to increase the background intensity of the characters on the GE OIT screen. All characters received by the GE OIT screen after this option has been set will be displayed in reverse video with the background set to a high intensity. This option can be disabled by calling function oit_char_att_1 with parameter set to DISABLE_VID.

**option: BK_DIM**
This option allows the user to decrease the background intensity of the characters on the GE OIT screen. All characters received by the GE OIT screen after this option has been set will be displayed in reverse video with the background set to dim intensity. This option can be disabled by calling function oit_char_att_1 with parameter set to DISABLE_VID.

Only the above options can be used with the function oit_char_att_1(). If the option specified is not one of the above, then the system will display an error message.
void oit_char_grph_1(int option, int parameter_1)

This is the second function associated with character attribute control on the GE OIT. This function gives the user access to the 33 supplemental graphic characters contained in the standard ASCII character set, the alternate graphic set and the preformed quad size characters. There exist special symbols for motor, valve, transformer, etc. which can be displayed easily by calling this function with the right option and parameter_1 setting. Refer to the Appendix A, B and C for an illustration of the graphic character sets 1 and 2 and the quad size characters which can be displayed on the GE OIT. All other video attributes may be used simultaneously with this function. Depending on the option set and the value of the parameter_1, the function enables or disables the appropriate graphic character set. Valid values for option and parameter_1, and the functionality achieved by them are explained below.

**option: GRAPHIC_SET1  parameter_1: ON, or OFF**

This option gives the user access to the 33 supplemental graphic characters contained in the standard ASCII character set. Once the function has been called with option set to GRAPHIC_SET1 and parameter_1 set to ON, graphic characters in the standard ASCII set can be displayed on the GE OIT. In order to display these graphic characters, the user should send the appropriate character associated with the
graphic symbol. Refer to the Appendix A Graphic Character Set 1 for the set of graphic characters that can be displayed on the GE OIT with this option. After this mode has been set, the equivalent graphic symbol of the character received by the GE OIT will be displayed on its screen. This mode can be disabled with option set to GRAPHIC_SET1 and parameter_1 set to OFF.

**option: GRAPHIC_SET2 parameter_1: ON, or OFF**

This option gives the user access to the alternate graphic character set shown in the Appendix B Graphic Set 2. Once this function has been called with option set to GRAPHIC_SET2 and parameter_1 set to ON, 95 displayable characters of the standard ASCII character set are substituted with the equivalent graphic symbol. In order to display these graphic characters, the user should send the appropriate character associated with the graphic symbol. Refer to the Appendix B Graphic Character Set 2 for the set of graphic characters that can be displayed on the GE OIT with this option. After this mode has been set, the equivalent graphic symbol of the character received by the GE OIT will be displayed on the screen. This mode can be disabled with option set to GRAPHIC_SET2 and parameter_1 set to OFF.
option: QUAD_SZ    parameter_1: ON, or OFF

This option gives the user access to the preformed quad size process control symbols called Industrial Icons shown in the Appendix C Quad Size Character Set. Apart from the special preformed quad size characters, other characters which can also be displayed in this mode are capitals (A-Z), numerals (1-9), and symbols (, . * + - = / ? ! SPACE). Once this function has been called with option set to QUAD_SZ and parameter_1 set to ON, the equivalent quad size symbol of the character received by the GE OIT is displayed on its screen. Refer to the Appendix C Quad Size Character Set for the set of quad size symbols that can be displayed on the GE OIT with this option. This mode can be disabled with option set to QUAD_SZ and parameter_1 set to OFF.

Only the above options and parameters can be used with the function oit_char_grph_1(). If the value of the parameters option and parameter_1 are not one of the above, then the system will display an error message.

3.3 Line Attribute Control

It is possible to set or reset additional attributes on a line by line basis on the GE OIT. These line attributes may be used with character attributes explained earlier, except that double wide character attributes are superseded by double wide line attribute. These attributes remain set on
the GE OIT until they are specifically cleared. There is only one function related with setting or resetting line attributes. This function has the character sequence 'ln' as a secondary element in its function name to distinguish it as a Line Attribute Control function. Thus the function name 'oit_ln_att_1' indicates that it is a function associated with line attribute control on the GE OIT.

oit_ln_att_1(int option)
This function lets a programmer set and reset certain character attributes on a line by line basis on the GE OIT. By making a call to this function, double height and, single height-double width characters can be set or reset on the GE OIT. Depending on the value of the option, the function sets or resets the line attributes. Valid values for the parameter option and the functionality achieved by it are explained below.

option: CLEAR_LNATT
This option disables all line attributes set previously on the GE OIT to a single height, single width. This command is useful in the beginning to clear all line attribute set previously on the terminal.

option: LN_DBL_HTWD_TOP
This option sets the current line the cursor is positioned at to double height character mode. The upper half of the
received characters are displayed on the GE OIT screen. Duplicate characters should be sent to the following line with option sent to LN_DBL_HTWD_BOT. This line attribute mode can be reset by setting the parameter option to CLEAR_LNATT.

**option: LN_DBL_HTWD_BOT**

This option sets the current line the cursor is positioned at to double height character mode. The lower half of the received characters are displayed on the GE OIT screen. Duplicate characters should have been sent to the previous line with option set to LN_DBL_HTWD_TOP. This line attribute can be reset by setting the parameter option to CLEAR_LNATT.

**option: LN_SNHT_DBWD**

This option sets the current line the cursor is positioned at to single height, double width character mode. Characters received by the GE OIT after this option has been set are displayed as such. This line attribute can be reset by setting the parameter option to CLEAR_LNATT.

Only the above options can be used with the function oit_ln_att_1(). If the option specified is not one of the above, then the system will display an error message.
3.4 Clock and Date Control

The GE OIT contains a battery-backed clock and calendar. This clock is always running even if the power is turned off. The GE OIT clock can be set or reset during terminal configuration, or by using special escape sequences sent from the host. When the clock and date display is enabled, the time and date are displayed on the status line, which is the lower right hand corner of the GE OIT screen.

Three functions have been defined which enable the programmer to set the battery-backed clock and calendar of the GE OIT. These functions have the character sequence 'cal' as a secondary element in their function name to distinguish them as Clock and Date Control functions. Thus the function name 'oit_cal_aaa_1' indicates that it is a function associated with clock and date control of the GE OIT.

int oit_cal_set_1(int option, char *cal_buffer)

This function allows the application programmer to set the time and date on the GE OIT. Depending on the value of the parameter 'option', either the time or the date is set. The value the time or date is set to, is specified in the second parameter of the function 'cal_buffer'. Valid values for the parameters (option and cal_buffer) and their functionality are explained below.
option: SET_TIME    cal_buffer: "HHMMSS"

This option allows the programmer to set the time on the GE OIT to a new value. The value to be set is specified in the parameter cal_buffer as a string with the format "HHMMSS", where "HH" indicates the hours, "MM" the minutes and "SS" the seconds. Three checks are performed on the value received in the parameter cal_buffer. The value in the "HH" (hour) portion of the cal_buffer should be between '00' and '23', and the value in the "MM" (minutes) and "SS" (seconds) portion of the cal_buffer should be between '01' and '59'.

option: SET_DATE    cal_buffer: "MMDDYY"

This option allows the programmer to set the date on the GE OIT to a new value. The value to be set is specified in the parameter cal_buffer as a string with the format "DDMMYY", where "DD" indicates the day, "MM" the month and "YY" the year. The following checks are performed on the value received in the parameter cal_buffer. The value in the "MM" (month) portion of the cal_buffer should be between '01' and '12', and the value in "DD" (day) portion of the cal_buffer should conform to the number of days in the month specified in the "MM" portion of the cal_buffer.

Only the above options can be used with the function oit_cal_set_1(). If the option specified is not one of the above, then the system will display an error message. The
function will return a zero integer value, if the format of the character string in the cal_buffer does not conform to the format stated here, and an integer one if the function is successful.

`void oit_cal_disp_1(int option, int parameter_1)`

This function allows the application programmer to enable or disable the time and date display on the GE OIT. Depending on the value of the parameters option and parameter_1, time or date display on the GE OIT is enabled or disabled. Valid parameter values for options and parameter_1 are explained below.

**option: DATE_DISPLAY**  **parameter_1: ON or OFF**

This option allows the programmer to enable or disable the date display on the GE OIT. A call such as `oit_cal_disp_1(DATE_DISPLAY, ON)` will enable the date display on the GE OIT, and the call `oit_cal_disp_1(DATE_DISPLAY, OFF)` will disable the date display on the status line of the GE OIT screen.

**option: TIME_DISPLAY**  **parameter_1: ON or OFF**

This option allows the programmer to enable or disable the time display on the GE OIT. A call such as `oit_cal_disp_1(TIME_DISPLAY, ON)` will enable the time display on the GE OIT, and the call
oit_cal_disp_1(TIME_DISPLAY, OFF) will disable the time display on the status line of the GE OIT screen.

Only the above options can be used with the function oit_cal_disp_1(). If the option specified is not one of the above, then the system will display an error message.

3.5 File Storage Control

The GE OIT has 32k bytes of user memory. The first 256 bytes of this user memory is used for function key program storage. This memory may be used to store user defined screens which are stored as files in the GE OIT's user memory. The memory itself is organized as files numbering from 0 through 254, and these files can be accessed in any order. Thus the memory can be used for storing up to 255 files, the size of the files being limited only by memory.

Files may be stored when the GE OIT is in the local or on-line mode. In the local mode, the screens are defined and stored using special escape sequences from the GE OIT keyboard whereas in the on-line mode, escape sequences for file definition and storage are received over the serial communications link from the host computer.

There exist special escape sequences for defining and saving new screens, appending to old screens, and displaying previously defined screens. Screen files consist of exact
sequence of characters entered or received during file storage. While defining screens in the local mode, the user should use absolute cursor positioning to position the cursor on the screen rather than using the cursor control keys.

While saving new screens, the GE OIT will retain any previously defined screens by that number until the new file has been successfully saved. This prevents old files from being lost in case of a power failure or other error. While new files are being saved or appended to, the cursor will be changed to form a half-block in the upper half of the character position to indicate that this operation is taking place. After the file has been successfully saved or appended to, the old file by that number is deleted, and the user memory on the OIT is consolidated to maximize the amount of remaining available space.

Six functions have been defined under the group FILE STORAGE CONTROL. These functions have the character sequence 'scr' as a secondary element in their function name to distinguish them as File Storage Control functions.

```c
void oit_scr_open_fl_1(int fileno)
```

This function must be called prior to defining new screens for storage. The screen being defined is identified from the parameter fileno. This function initiates the process
by which new screen files are stored. After calling this
function, a series of other function calls should be made to
position cursor, output characters, draw symbols, set
attributes etc. which are required for the screen
definition.

Once the screen has been designed, that is screen design
data has been input, the function oit_scr_close_1 should be
called. This function signals the end of the file storage.
If oit_scr_close_1 is called without intermediate screen
data, any existing old file by the same file number will be
deleted. It is important to note that screens created
while in one terminal configuration may be displayed
differently if the terminal configuration is different when
they are recalled. The following general format is
recommended for defining and storing screen files.

oit_scr_open_1(fileno)
oit_edt_disp_1(CLR_SCR)
oit_ln_att_1(CLEAR_LNATT)

enter screen data
.
.
.
oit_scr_close_1()
void oit_scr_close_1(void)

This function signals the end of file storage. This function should be called after screen data has been defined or after an existing file has been appended to. Every call to the function oit_scr_open_1 or the function oit_scr_append_1 should be followed by screen design data which should be followed by the function scr_close_fl_1 to indicate the end of file storage.

void oit_scr_append_1(int fileno)

Existing files can be modified and added to using this function. The parameter fileno identifies the file for which the append operation is being performed on. This function call should be followed by screen design data and finally a call to the function oit_scr_close_1() should be made, to signal the end of the append file mode.

void oit_scr_display_1(int fileno)

This function displays the screen file previously stored in the GE OIT's memory using the function oit_scr_open_1() on the GE OIT screen. The screen file which is displayed after a call to this function has been made could have been created in the local or on-line mode of the GE OIT. In the local mode of operation, the screen data is entered from the GE OIT keyboard. In the on-line mode of operation, the screen file is created using the function oit_scr_open_1.
The screen file displayed on the GE OIT screen is identified by the parameter fileno. The parameter fileno should have a value between 0 and 254.

void oit_scr_reinit_1(void)

If at any time the CMOS battery back-up is lost the file storage area should be reinitialized. This function accomplishes this purpose. This function also clears the function key program storage memory, which is located in the first 256 bytes of the GE OIT’s user memory.

3.6 Function Key Programming

The first 256 bytes of user memory are used for Function Key Program storage. It is possible to program the function keys to accomplish specific purposes. The length of the function key program should be limited to 16 characters. The program can be displayable data, or any other valid escape sequence.

Function key programming can be accomplished in the local or on-line mode. The user can configure the GE OIT to local mode of operation and perform function key programming entering characters from the GE OIT keyboard or the programming can be done on-line using data transmitted from the host computer.
The user can enter up to 16 characters and identify them to be associated with a particular function key. Thus the host can transmit the escape sequence which identifies the key to the GE OIT and the user defined program associated with the function key is executed. Two functions have been defined which enable the programmer to perform function key programming from the host computer. These functions have the character sequence ‘fnc’ as the secondary element in their function name to distinguish them as Function Key Programming functions.

void oit_fnc_start_1(int fnkey)

This function initiates the start of a function key program. The function key for which the program is being defined is identified by the parameter fnkey which should have a value between 1 and 15. This function should be followed by the escape sequence or data which constitutes the function key program and finally followed by the function oit_fnc_stop_1() which indicates the end of the function key program definition.

void oit_fnc_stop_1(void)

This function signals the end of a function key program definition.
3.7 Configuration Control

The GE OIT has several different options regarding data communication format and operating features. Several of these operating modes namely the baud rate, parity, communication type - point to point or multidrop, status line setting, and local/on-line mode of operation can only be set in the local mode of operation of the GE OIT. Apart from the above modes, the remaining of the configuration modes namely, cursor type - block or blinking, auto line feed option, full or half duplex mode, VT52/ANSI escape sequence, auto wrap option, enable/disable keyboard, and enable/disable display are software programmable. Three functions have been defined which enable the programmer to configure the GE OIT to different options. These functions have the character sequence 'cfg' as the secondary element in their function name to distinguish them as Configuration Control functions.

void oit_cfg_model_1(int option, int parameter_1)

Certain operating modes like cursor type, auto line feed, auto carriage return, blink cursor, Full/Half duplex mode, and enable/disable cursor keys can be set on the OIT using this function. The configuration mode being set to depends on the value of the parameters option and parameter_1.
Valid values for the parameters (option and parameter_1) and their functionality are given below.

**option: BLOCK_CURSOR**  
**parameter_1: ON or OFF**  
This option allows the programmer to control the type of cursor being displayed on the GE OIT - block or underscore. If the parameter option is set to BLOCK_CURSOR and parameter_1 set to ON, then the cursor displayed on the GE OIT screen is of block type. If the parameter option is set to BLOCK_CURSOR and parameter_1 set to OFF, then the cursor displayed on the GE OIT screen is of underscore type.

**option: CURSOR_DISP**  
**parameter_1: ON or OFF**  
This option allows the programmer to enable or disable the cursor display. If the parameter option is set to CURSOR_DISP and parameter_1 set to ON, then the cursor of the type set previously is displayed on the GE OIT screen. If the parameter option is set to CURSOR_DISP and parameter_1 set to OFF, then no cursor is displayed on the GE OIT screen.

**option: AUTO_LF**  
**parameter_1: ON or OFF**  
This option when set generates a automatic linefeed on the receipt of the carriage return by the GE OIT.
**option: AUTO_CR**

**parameter_1: ON or OFF**

This option when set generates a automatic carriage return on the receipt of a line feed character by the GE OIT.

**option: BLNK_CURSOR**

**parameter_1: ON or OFF**

This option allows the programmer to set a blinking or a non-blinking cursor on the screen. If the parameter option is set to BLNK_CURSOR and parameter_1 set to ON, then a blinking cursor of the type set previously is displayed on the GE OIT screen. If the parameter option is set to BLNK_CURSOR and parameter_1 set to OFF, then a non-blinking cursor (of the type set previously) is displayed on the GE OIT screen.

**option: SEL_FULL_DPX**

**parameter_1: ON or OFF**

This option allows the programmer select the half-duplex mode or the full-duplex mode of operation of the GE OIT. If the parameter option is set to SEL_FULL_DPX and parameter_1 set to ON, then the GE OIT will be configured to operate in the full-duplex mode. However, if the parameter option is set to SEL_FULL_DPX and parameter_1 set to OFF, then the GE OIT will be configured to operate in the half-duplex mode.
option: DIS_CUR_KEY    parameter_1: ON or OFF
This option allows the programmer to enable or disable the
cursor keys on the GE OIT keyboard. If the parameter option
is set to DIS_CUR_KEY and parameter_1 is set to ON, then the
cursor keys on the GE OIT keyboard will have no effect in
the local mode, and in the on-line mode nothing will be
transmitted over the communications link. If the parameter
option is set to DIS_CUR_KEY and parameter_1 set to OFF,
then the cursor keys of the GE OIT keyboard will be enabled.

Only the above options can be used with the function
oit_cfg_mode1_1(). If the option specified is not one of
the above, then the system will display an error message.

void oit_cfg_mode2_1(int option,int parameter_1)
Certain operating modes like escape sequence type,
enable/disable display and wrap on/off can be set on the OIT
using this function. The configuration mode being set to
depends on the value of the parameters option and
parameter_1. Valid values for the parameters (option and
parameter_1) and their functionality are explained below.

option: ANSI_ESCAPE_SEQ parameter_1: ON or OFF
This option allows the programmer to set the GE OIT to
follow either ANSIx3.64 or DEC VT52 escape sequences. If
the parameter option is set to ANSI_ESCAPE_SEQ and parameter_1
is set to ON, then the GE OIT is configured to follow the
ANSIx3.64 escape sequences and if the parameter_1 is set to OFF, then the GE OIT is configured to follow the DEC VT52 escape sequences.

**option: WRAP_AT_EOL**  
**parameter_1: ON or OFF**

This option configures the GE OIT to send a automatic carriage return and line feed when a character is displayed on the 80th column. If the parameter option is set to WRAP_AT_EOL and parameter_1 is set to ON, then the GE OIT is configured to send a automatic carriage return and line feed when a character is displayed on the 80th column. If the parameter_1 is set to OFF, then the characters received by the GE OIT do not wrap when the end of line, that is the 80th column is reached.

Only the above options can be used with the function oit_cfg_mode2_1(). If the option specified is not one of the above, then the system will display an error message.

**void oit_cfg_mode3_1(int option, int parameter_1)**

Certain configuration modes like enable/disable keyboard and enter/exit insert character mode can be set on the OIT using this function. The configuration mode being set to depends on the value of the parameters option and parameter_1. Valid values for the parameters (option and parameter_1) and their functionality are explained below.
option: ENA_KYBD  

This option allows the programmer to enable or disable the GE OIT keyboard. If the parameter option is set to ENA_KYBD and parameter_1 is set to ON, then the GE OIT keyboard will have no effect in the local mode, and in the on-line mode nothing will be transmitted over the communications link. If the parameter option is set to ENA_KYBD and parameter_1 set to OFF, then the keyboard attached to the GE OIT will be enabled.

option: ENTR_INSERT  

This option configures the GE OIT to enter or exit the insert character mode. If the parameter option is set to ENTR_INSERT and parameter_1 is set to ON, then the GE OIT is configured to be in the insert character mode, namely whenever a character is received by the GE OIT, it will push the existing characters to the right of the cursor, one place to the right and if the parameter_1 is set to OFF, then the GE OIT will exit the insert character mode.

Only the above options can be used with the function oit_cfg_mode3_1(). If the option specified is not one of the above, then the system will display an error message.
3.8 Cursor Control

The GE OIT cursor movement is software controllable. Thus the application programmer can position the cursor anywhere on the screen, save and restore the cursor position, and also report the current cursor position to the host computer. Three functions have been defined which enable the programmer to control the cursor movement on the GE OIT screen. These functions have the character sequence 'cur' as the secondary element in their function name to distinguish them as Cursor Control functions.

\texttt{void oit_cur_mov_1(int option, int parameter_1)}

The cursor location on the GE OIT screen is software programmable. This function enables the programmer to perform relative cursor positioning. The cursor can be moved to a new location which is either above or below, or to the right or left of the current cursor position. In addition to the above movements, the cursor can also be moved to the HOME position which is the first column and first row of the GE OIT screen. Valid values for the parameters (option and parameter_1) and their functionality are explained below.
option: CURSOR_HOME      parameter_1: ignored
This option, when set positions the cursor in the HOME position, which is the first row and first column of the GE OIT screen.

option: CURSOR_UP       parameter_1: number of lines
This option, when set moves the cursor up from its current location. The number of lines the cursor moves by depends on the value specified in the parameter parameter_1.

option: CURSOR_DOWN     parameter_1: number of lines
This option, when set moves the cursor down from its current location. The number of lines the cursor moves by depends on the value specified in the parameter parameter_1.

option: CURSOR_FORWARD  parameter_1: number of columns
This option, when set moves the cursor right from its current location. The number of positions the cursor moves by depends on the value specified in the parameter parameter_1.

option: CURSOR_BACKWARD parameter_1: number of columns
This option, when set moves the cursor left from its current location. The number of positions the cursor moves by depends on the value specified in the parameter parameter_1.
Only the above options can be used with the function oit_cur_mov_1(). If the option specified is not one of the above, then the system will display an error message.

```c
void oit_cur_posn_1(int row_no, int col_no)
```

The cursor location on the GE OIT screen is software programmable. This function enables the programmer to perform absolute cursor positioning. The cursor can be moved to a new location which is specified in terms of a row and column number. If the value specified for the row does not exist, then the cursor will remain in the current row. If the value specified for the column does not exist, then the cursor will move to the rightmost column of the selected row.

```c
void oit_cur_svrs_1(int option)
```

The current cursor position can be saved and restored to at a later time. Valid values for the parameter option and its functionality is explained below.

- **option: SAVE**
  
  This option saves the current cursor position so that the cursor can be returned to it later using the option RESTORE.

- **option: RESTORE**
  
  This option restores the cursor to the position saved last with the option SAVE.
Only the above options can be used with the function oit_cur_mov_1(). If the option specified is not one of the above, then the system will display an error message.

3.9 **Erase and Edit Control**

Editing of the GE OIT screen is possible using software control. The programmer can have control over the contents of the display and perform editing from his or her host computer. The application programmer can clear screen, insert and delete lines, insert and delete characters, and erase lines. Three functions have been defined which enable the programmer to perform erasing and editing functions from the host computer. These functions have the character sequence 'edt' as the secondary element in their function name to distinguish them as Erase and Edit functions.

```c
void oit_edt_disp_1(int option)
```

This function erases information displayed on the GE OIT screen. Depending on the input parameter, either the entire screen is erased, or current cursor location to the end of the screen is erased, or beginning of screen to the current cursor location is erased. Valid values for the parameter option and its functionality are explained below.

**option: CLR_SCR**

The entire OIT screen is filled with spaces.
option: CLR_SCR_TO_CUR
The GE OIT screen is erased from the beginning to the current cursor location.

option: CLR_SCR_FR_CUR
The OIT screen is erased from the current cursor location to the end of screen.

Only the above options can be used with the function oit_edt_disp_1(). If the option specified is not one of the above, then the system will display an error message.

void oit_edt_ln_1(int option)
This function erases information displayed on the current row of the cursor. Depending on the input parameter, either the entire line is erased, or current cursor location to the end of the line is erased, or beginning of the line to the current cursor location is erased. Valid values for the parameter option and its functionality are explained below.

option: CLR_LINE
The entire line, including the current cursor position is erased.

option: CLR_LINE_TO_CUR
Erases information displayed from the beginning of the line to the current cursor position.
**option: CLR_LINE_FR_CUR**

Erases from the current cursor location to the end of the line.

Only the above options can be used with the function `oit_edt_ln_1()`. If the option specified is not one of the above, then the system will display an error message.

**void oit_edt_char_1(int option, int parameter_1)**

This function deletes characters and lines of text displayed on the GE OIT screen. The value of the parameter `parameter_1` specifies the number of lines or characters to be deleted. Valid values for the parameters (option and `parameter_1`) and their functionality are explained below.

**option: DEL_CHAR    parameter_1: number of characters**

The number of characters to be deleted from the current line is specified in the parameter `parameter_1`. Any existing text to the right of the cursor is shifted one character to the left.

**option: DEL_LINE    parameter_1: number of lines**

The number of lines of text to be deleted from the current line is specified in the parameter `parameter_1`. All the lines following the current line are moved up and blank lines are added above the status line.
Only the above options can be used with the function oit_edt_char_1(). If the option specified is not one of the above, then the system will display an error message.

3.10 Graphics Display

Five functions have been included in this group, and these functions are intended to make screen programming easier for the user. The applications programmer can draw bar graphs, change attributes in a particular region, draw a box, define a scroll region, or a solid region easily by using these functions. These functions have the character sequence 'grph' as the secondary element in their function name to distinguish them as functions belonging to the Graphics Display group.

void oit_grph_setatt_1(int rowno, int colno)

This function changes the character attributes in a particular region, without affecting the text being displayed. The user should first call the function to change the character attributes (any of the functions belonging to the character attribute control can be used) and then position the cursor in the lower left corner of the required region and then make a call this function. The parameters rowno is the number of rows and colno is the number of columns in the region.
void oit_grph_fillrg_1(int rowno, int colno, char fill_char)

This function fills a specified region with a particular character. The character can be a space, a graphic character, or any other displayable character. The user should first position the cursor in the lower left corner of the required region and then make a call this function. The parameters rowno is the number of rows, and colno is the number of columns in the region. The parameter fill_char is the character to be used for filling the region. This function is useful for drawing vertical or horizontal lines as well as erasing a box or filling a box.

void oit_grph_drbox_1(int rowno, int colno)

This function can be used to draw a box outline for a specified region. The characters within the box are unaffected. The user should first position the cursor in the lower left corner of the required region and then make a call this function. The parameters rowno is the number of rows, and colno is the number of columns in the region.

void oit_grph_bar_1
(int option, int cell_length, int cell_frc, int num_cells)

This function can be used to generate horizontal and vertical bar graphs on the GE OIT screen. Each call to the function will generate a bar of the length specified in the
two parameters cell_length and cell_frc. The parameter cell_length represents the total number of character cells to be used for the bar, the parameter cell_frc represents the percentage of the fractional cell to which the bar should be extended, and the parameter num_cells represents the maximum number of cells that a bar can have. Finally the parameter option specifies the type of bar graph to be displayed, that is, horizontal or vertical. The value in the parameter call_frc should be between 0 and 99. The valid options and their explanations are given below.

**option: HORIZONTAL_BAR**

This option should be specified when the user wishes to draw horizontal bar graphs on the OIT display screen. When this option is used, the value specified in the parameter, num_cells, should be between 1 and 25. Horizontal bar are displayed to the nearest 1/8th of cell, resolution for 80 columns being 1:640.

**option: VERTICAL_BAR**

This option should be specified when the user wishes to draw vertical bar graphs on the OIT display screen. When this option is used, the value specified in the parameter, num_cells, should be between 1 and 80. Vertical bars are displayed to the nearest 1/10th of cell, resolution for 24 lines being 1:240.
void oit_grph_sp1scr_1(int rowno1, int rowno2)

This function defines a scrolling region on the GE OIT screen. This region is defined in terms of a top row - rowno1 and a bottom row - rowno2. After this option has been set, any data sent to the GE OIT display screen will now appear within the specified region. If the data does not fit into the region, then it will scroll within it. If the cursor is currently positioned within the scroll region, then the only mechanism by which it can moved to a new position which is outside the region, is by calling the function oit_cur_posn_1(). As mentioned earlier, the parameter rowno1 defines the top row of the scroll region, and the parameter rowno2 defines the bottom row of the scroll region.

3.11 Misc Function

There exist one miscellaneous function which is important but does not fall under any special group. This is the function oit_send_rcv_1.

All the functions explained earlier make use of the function oit_send_rcv_1 to send or receive data from the GE OIT. This function (oit_send_rcv_1) sends and receives data from the OIT_CONTROL which is an autoload program on Model of the Communications Systems/2 Network. The OIT_CONTROL program
is explained in Chapter 5. It is the controlling program for the GE OIT, for it is the only mechanism by which the GE OIT can send or receive data from any application program on the network. Any program sitting on any node of the communications network would have to communicate with the OIT_CONTROL in order to display or receive any data from the GE OIT. This communication link between the application program and the OIT_CONTROL is provided by the function oit_send_rcv_1.

**oit_send_rcv_1 (int option, char *buffer, int buf_len)**

All the functions explained earlier make use of this function to send or receive data from the program OIT_CONTROL. This function makes use of API calls to receive data from the OIT_CONTROL and send data to the OIT_CONTROL. The parameter buffer points to the address of the character string to be sent or received and its length should be specified in the parameter buf_len. Valid values for the parameter option are given below.

**option: SEND**

This is the only mechanism by which the programmer can send data to GE OIT. When the function is used to send data to the GE OIT, then the option is set to SEND. The character string to be sent is specified in the parameter buffer, and its length is given in the parameter buf_len.
**option: RECEIVE**

This is the only mechanism by which the programmer can receive data from GE OIT. When the function is used to receive data from the GE OIT, then the option is set to RECEIVE. The length of the character string to be received is specified in the parameter buf_len and the character string input by the operator from the GE OIT keyboard is returned in the parameter buffer.
CHAPTER 4. I/O CONTROL MODULE

4.1 Specifications

The I/O control module was defined to be an autoload program. It came into existence only when it had pending requests or messages on its ACB queue. This program sits on a node which is connected to the GE OIT terminal through a serial I/O interface at RS-232-C signal level.

This program checked for two types of CS2 requests on its ACB queue - CS2_SEND() and CS2 SNDREC(). The request CS2_SEND was used by the library routines to send messages to the GE OIT through the I/O control module, and the request CS2 SNDREC was used by the library routines to receive data from the GE OIT through the I/O control module. When either of the above requests was received by the I/O control module, the module came alive and performed the required operations.

The function getmsg() receives pending requests and messages from its ACB queue. It issues the API call CS2 RECEIVE and receives the awaiting messages in its receive structure. The order of the received messages is based upon the priority set by the application program issuing the request. The I/O control module functions by receiving requests from its ACB Queue. It processes the request depending on
whether data needs to be output to the GE OIT or input from the GE OIT. If data needs to be output, the function `drv_out_data()` is called. This function accesses its queue for the data, and outputs the required string of data (control data or display data) on the GE OIT. If data needs to be input, then the function `drv_in_data()` is called. This function takes input data from the GE OIT, which is entered by the operator, and stores it in its queue, to be retrieved later by the originator of the request.

All the initializations required for the COM1 port on Model of the network are handled by the function `init_port()`. The required parameters for serial port initialization like baud rate, parity, and number of data bits are set using the functions `DosDevIOCtl()`, `DosWrite()`, `DosRead()`, and `DosBufReset()` provided in the OS/2 library.

The function `check_pend_req()` is used check the ACB queue of the I/O control module for pending requests, and if no requests are pending then the program is purged from the control block pool. The API call `CS2_COND_PURGE` is issued by this function to determine if there are any pending requests in its ACB queue. This API request purges the program’s ACB from the control block pool, only if there are
no pending requests. However, if any requests are pending, then the ACB is not purged.
CHAPTER 5. DEMONSTRATION PROGRAM

5.1 Overview

The demonstration program was loaded at Node2 of the network. It provides the future user of the system with sample code and shows the correct usage of the functions provided in the library. It also shows how the future user should initialize the required structures for the DAE API calls. Additionally, it demonstrates the capability of the library functions.

This chapter contains the details of the DAE environment setup, namely the control blocks and CS2 requests. The control blocks that need to be setup by the programmer and their corresponding field values are given. All the CS2 requests that were used in the program are explained. Finally, the processing details of the program is provided.

5.2 Required Control Blocks

The control blocks that need to be defined on a DAE node at which the demonstration program is loaded is given in Table 5.1. There were six control blocks defined on Node2 of the network. These control blocks represent the system resources and provide the required information about the resources to the DAE. An Application Control Block (ACB)
Table 5.1 Control Blocks on Node2

<table>
<thead>
<tr>
<th>Control Blocks on Node2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ACB</strong> - Application Control Block</td>
</tr>
<tr>
<td>Name of the control block: DEMO.CS2NODE</td>
</tr>
<tr>
<td><strong>CCB</strong> - Communications Control Block (Local)</td>
</tr>
<tr>
<td>Name of the control block: NODE2.CS2AP000</td>
</tr>
<tr>
<td><strong>CCB</strong> - Communications Control Block (Remote)</td>
</tr>
<tr>
<td>Name of the control block: NODE1.CS2AP000</td>
</tr>
<tr>
<td><strong>CACB</strong> - Communications Access Control Block</td>
</tr>
<tr>
<td>Name of the control block: CS2AP000.CS2NODE</td>
</tr>
<tr>
<td><strong>ANCB</strong> - Alias Name Control Block</td>
</tr>
<tr>
<td>Name of the control block: OIT_CTRL.CS2NODE</td>
</tr>
<tr>
<td><strong>RCB</strong> - Remote Control Block</td>
</tr>
<tr>
<td>Name of the control block: OIT_CTRL.NODE1</td>
</tr>
</tbody>
</table>
was setup for the demonstration program. The fields that were defined for this control block are listed in Table 5.2. This ACB served to identify the demonstration program as a resource of the DAE. Thus, the demonstration program could avail the DAE services, make API calls, and use the network communications component to communicate with the I/O control module on Node1 of the network.

Two Communications Control Blocks (CCBs) were setup on Node2. One represented the local node (Node2), and the other identified the remote node (Node1) with which the local node (Node2) communicated. The fields that were defined for these CCBs are listed in Table 5.2.

DAE requires a Communications Access Control Block (CACB) to be defined on each node on a communications network. A CACB was defined on Node2 of the network. This control block represented the communications access line for the node. The field values for this control block are listed in Table 5.2.

A Remote Control Block (RCB) was also defined. This RCB identified a control block on a remote node with which the local node communicated. Finally, an Alias Name Control Block was defined on Node2. This identified the location of the physical resource for the alias "OIT_CONTROL". The field values for this control block are listed in Table 5.2.
### Table 5.2 Control Blocks Details for Node2

**Control Blocks on Node2**

**ACB - Application Control Block**
Name of the program: DEMO.CS2NODE  
Path to Program: d:\c600\proj\src\demo.c  
Load during initialization: Off

**CCB - Communications Control Block (Local)**
Name of the control block: NODE2.CS2AP000

**CCB - Communications Control Block (Remote)**
Name of the control block: NODE1.CS2AP000

**CACB - Communications Access Control Block**
Name of the control block: CS2AP000.CS2NODE

**ANCB - Alias Name Control Block**
Name of the control block: "OIT_CONTROL"  
Physical Resource Name: OIT_CTRL.NODE1

**RCB - Remote Control Block**
Name of the control block: OIT_CTRL.NODE1  
Controlling CCB: NODE1.CS2AP000
5.3 Processing

The demonstration program is a menu driven program, and depending on the user input, either displays the appropriate screen, or performs an OIT operation. The menu has 2 levels. The highest level is the main menu where the user selects the type of operation to be performed. Depending on the operation selected by the user, a suitable sub-menu is displayed. The sub-menu displays a list of valid operations which can be performed at the OIT and depending on the value input by the user, an appropriate function is called. For example, if the user selects to configure the GE OIT, then a sub-menu with a list of options is displayed. The user can select: Half Duplex Mode, Full Duplex Mode, ANSI Escape Sequence Mode, VT52 Escape Sequence Mode, Hardware Handshake Mode, Software Handshake Mode or exit back to the main menu.

At the start of the program, the necessary API calls - CS2_INIT() and CS2_INITIALIZE_RES() are made. The CS2_INIT ensures that the DAE is initialized and is running properly. Since the demonstration program is not started by Communications System, CS2_INITIALIZE_RES() is then called. This API call provides a mechanism for the program to use Communications System's services. After the above operations, a main menu is displayed. Here the user can select the type of operation he or she wishes to perform:
Configure the OIT, Perform Screen Operations, Graphics Display, or Function Key Operations. This menu is illustrated in Figure 5.3. Selecting any of the above options, brings up the suitable sub-menu at the lower level. The user can choose to exit the program by choosing option 5. When the user selects to configure the OIT, then the function disp_cfg_menu() is called. This sub-menu is illustrated in Figure 5.4. Here the following options are presented: Half Duplex Mode, Full Duplex Mode, ANSI Escape Sequence Mode, VT52 Escape Sequence Mode, Software Handshake Mode and Hardware Handshake Mode. Here the use of the following two functions oit_cfg_mode1_1() and oit_cfg_mode2_1, with different parameters is demonstrated. The user can exit this sub-menu by choosing option 7, and this will take him or her back to the main menu. When the user chooses to perform screen operations, then the function disp_scr_menu() is called. This function displays the screen operations sub-menu. This sub-menu is illustrated in Figure 5.5. Here the user is presented with the following options: Display Sample Screen, and, Receive Sample Data from the OIT. If the user selects the first option, then the function oit_scr_display_1() is called, and, this function displays a sample screen which had been programmed previously. This sample screen had been created using the function pgm_scr1(). This option illustrates how data can
MAIN MENU

1. Configuring OIT
2. Screen Operations
3. Graphics Display
4. Function Key Operations
5. Exit

Enter Option:

Figure 5.3 Main Menu Screen
CONFIGURE THE OIT

1. Half Duplex Mode
2. Full Duplex Mode
3. ANSI Escape Sequence Mode
4. VT52 Escape Sequence Mode
5. Software Handshake Mode
6. Hardware Handshake Mode
7. Exit to Main Menu

Enter Option:

Figure 5.4 Configure Menu Screen
SCREEN OPERATIONS

1. Display Sample Screen
2. Receive Sample Data from OIT

3 Exit to Main Menu

Enter Option:

Figure 5.5 Screen Operations Menu Screen
be displayed on the GE OIT screen from Node2 of the communications network. Table 5.6 contains the list of functions with parameters, which were used in programming this screen. If the user selects the second option, then the function oit_scr_display_1() is again is called, and this function displays a second sample screen which had also been programmed previously. This screen had been created using the function pgm_scr3(). Table 5.7 contains the list of functions with parameters, which were used in the second option. This option demonstrates how the data input by a user at the GE OIT keyboard can be received at Node2. After the sample screen is displayed on the GE OIT display screen, the program waits for operator input by calling the function rcv_data(). This function calls the library function oit_send_rcv_1() with the first parameter being RECEIVE, and receives the data entered by the operator at the GE OIT keyboard and displays it on Node2. The user can exit this sub-menu by choosing option 3 and this will take him or her back to the main menu.

The third option demonstrates the purpose of the graphic and character attribute functions provided in the library, and also shows how they can be used to program effective displays on the GE OIT screen. When the user chooses option 3 from the main menu, the function disp_grph_menu(). This function displays the sub-menu illustrated in Figure 5.8.
Table 5.6 Functions used in programming screen 1

oit_scr_open_1(1)
oit_edt_disp_1(CLr_SCR)
oit_char_att_1(DISABLE_VID)
oit_cur_posn_1()
oit_grph_drbox_1()
oit_char_att_1(HGlt_VID)
oit_char_att_1(REV_VID)
oit_send_rcv_1()
oit_ln_att_1(CLEAR_LNATT)
oit_scr_close_1()
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>oit_scr_display_1()</td>
</tr>
<tr>
<td>oit_cfg_model_1(DIS_CUR_KEY, ON)</td>
</tr>
<tr>
<td>oit_cur_posn_1()</td>
</tr>
<tr>
<td>oit_send_rcv_1()</td>
</tr>
<tr>
<td>oit_scr_open_1(1)</td>
</tr>
<tr>
<td>oit_edt_disp_1(CLR_SCR)</td>
</tr>
<tr>
<td>oit_char_att_1(DISABLE_VID)</td>
</tr>
<tr>
<td>oit_cur_posn_1()</td>
</tr>
<tr>
<td>oit_grph_drbox_1()</td>
</tr>
<tr>
<td>oit_char_att_1(HGLT_VID)</td>
</tr>
<tr>
<td>oit_char_att_1(REV_VID)</td>
</tr>
<tr>
<td>oit_ln_att_1(LN_SNHT_DBWD)</td>
</tr>
<tr>
<td>oit_ln_att_1(CLEAR_LNATT)</td>
</tr>
<tr>
<td>oit_scr_close_1()</td>
</tr>
<tr>
<td>oitCfg_model_1(DIS_CUR_KEY, OFF)</td>
</tr>
</tbody>
</table>
GRAPHICS DISPLAY
1. Horizontal Bar Graph Display
2. Vertical Bar Graph Display
3. Flexible Manf. Cell Layout
4. Exit to Main Menu

Enter Option:

Figure 5.8 Graphics Display Menu Screen
Here the user is presented with the following options: Horizontal Bar Graph Display, Vertical Bar Graph Display and Flexible Manufacturing Cell Layout. Table 5.9 contains the list of all the functions which were used in creating the two types of bar graphs and Table 5.10 contains the list of all the functions which were used in creating the Flexible Manufacturing Cell Layout.

The fourth option demonstrates the use of the function key operations on the GE OIT. This sub-menu is illustrated in the Figure 5.11. Here the user is presented with the following two options: Program Function Key, and Execute Function Key Program. The use of the four library functions oit_fnc_start_1(), oit_fnc_stop_1(), oit_send_rcv_1() and oit_edt_disp_1() are demonstrated in this option. The user can choose to exit back to the main menu by choosing option 3.

If the user chooses to exit the program from the main menu, then the function exit_pgm() is called. This function calls the API function CS2_STOP_RES(). This API call stops the demonstration program’s Application Control Block (ACB) and prevents further requests from getting queued to it. The return codes in the Request Data Structure (RDS) are now checked to ensure that the ACB has been stopped successfully. Finally the user exits the system.
Table 5.9 Functions Used in Bar Graph Display

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>oit_cur_posn_1()</td>
</tr>
<tr>
<td>oit_send_rcv_1()</td>
</tr>
<tr>
<td>oit_scr_open_1(1)</td>
</tr>
<tr>
<td>oit_edt_disp_1(CLR_SCR)</td>
</tr>
<tr>
<td>oit_char_att_1(DISABLEVID)</td>
</tr>
<tr>
<td>oit_grph_bar_1(HORIZONTAL_BAR)</td>
</tr>
<tr>
<td>oit_grph_bar_1(VERTICAL_BAR)</td>
</tr>
<tr>
<td>oit_grph_drbox_1()</td>
</tr>
<tr>
<td>oit_scr_display_1()</td>
</tr>
<tr>
<td>oit_scr_close_1()</td>
</tr>
</tbody>
</table>
Table 5.10 Functions used in Fl. Manf. Layout Option

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oit_cur_posn_1()</td>
<td></td>
</tr>
<tr>
<td>oit_send_rcv_1()</td>
<td></td>
</tr>
<tr>
<td>oit_scr_open_1(1)</td>
<td></td>
</tr>
<tr>
<td>oit_edt_disp_1(CL_R_SCR)</td>
<td></td>
</tr>
<tr>
<td>oit_char_att_1(DISABLE_VID)</td>
<td></td>
</tr>
<tr>
<td>oit_grph_drbox_1()</td>
<td></td>
</tr>
<tr>
<td>oit_char_att_1(GRAPHIC_SET1, ON)</td>
<td></td>
</tr>
<tr>
<td>oit_char_att_1(GRAPHIC_SET1, OFF)</td>
<td></td>
</tr>
<tr>
<td>oit_char_att_1(GRAPHIC_SET2, ON)</td>
<td></td>
</tr>
<tr>
<td>oit_char_att_1(GRAPHIC_SET2, OFF)</td>
<td></td>
</tr>
<tr>
<td>oit_scr_display_1()</td>
<td></td>
</tr>
<tr>
<td>oit_scr_close_1()</td>
<td></td>
</tr>
</tbody>
</table>
FUNCTION KEY OPERATIONS

1. Program a Function Key
2. Execute a Function Key Pgm
3. Exit to Main Menu

Enter Option:

Figure 5.11 Function Key Operations Menu Screen
CHAPTER 6. REQUIRED HARDWARE AND SOFTWARE PRODUCTS

The hardware and software products required for the system implementation are outlined in this chapter.

6.1 Hardware Requirements

There are four hardware products required for the implementation of the project: System Unit, GE Operator Interface Terminal, IBM Token Ring Adapter, and the Network Communications.

Two system units (IBM Industrial Computer 7552) were installed and connected via a Token-Ring communications network. These units constituted the two nodes Node1 and Node2 of the DAE network.

A GE OIT was connected to one of the system units through a serial I/O interface at an RS-232-C signal level. All communications between the GE OIT and the system unit was handled by this interface.

An IBM Token-Ring Network adapter II was installed on the two system units and connected through a network. The above adapter was required because the system unit used the network REM of DAE to access the network.
The network communication component of DAE transfers data between the different nodes on a network. The network communications component, is the communications Resource Execution Module (REM). There must be a communications REM loaded at each node of the DAE network. The APPC REM of DAE was selected for communications, and its details are listed below.

<table>
<thead>
<tr>
<th>Communications REM</th>
<th>Communications Protocol</th>
<th>Network Adapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS2APR00</td>
<td>APPC</td>
<td>IBM Token-Ring Adapter II</td>
</tr>
</tbody>
</table>

6.2 Software Requirements

There are three software products required to implement the system developed in this project: the OS/2 operating system, Language Support (MicroSoft C), and the Communications System/2 (DAE).

Communications Systems/2 (DAE) requires that the operating system OS/2 be installed on the system unit. The OS/2 version installed on the two system units was IBM Operating System/2 Extended Version 1.2. Also, the OS/2 Communications Manager was installed at each node of the network. The OS/2 Communications Manager provides API services that enable application programs to communicate with other computers on a network. The APPC services of the Communications Manager were selected for communication. The
DAE APPC Resource Execution Module (REM) which was selected, supports the above protocol of the Communications Manager.

The software development was done using the Microsoft C Version 6.00A Optimizing Compiler. The DAE API calls support the above compiler.

The software development was done under the DAE environment. DAE was loaded at each node of the communications network. The DAE programming environment and its control blocks are described in the following section.

6.3 DAE Programming Environment

The Distributed Automation Edition environment (DAE environment) Version 2.0, was used for developing the system. The Communications System/2 (or DAE) comes with two network Resource Execution Modules (REMs) namely the APPC and NETBIOS. These REMs, let the Communication System/2 (CS/2) nodes communicate over a network. The APPC REM was selected for communication purposes. This REM lets the CS/2 nodes communicate over the IBM Token-Ring Network using the OS/2 Communications Manager SNA APPC protocol.

As mentioned earlier, all existing resources on the DAE network should be defined. Thus several control blocks were setup on each node of the DAE network. An Application Control Block was set up for the I/O Control Module —
OIT_CTRL.CS2NODE on Node1 of the network, and another one for the demonstration program - DEMO.CS2NODE was defined on Node2. An Alias Name Control Block (ANCB) was also set up for the program OIT_CTRL.CS2NODE. Apart from the above control blocks, several Communications Control Blocks (CCBs), Communication Access Control Blocks (CACBs) and Remote Control Blocks (RCB) were also defined on the DAE nodes. The details of all the control blocks on each node of the network is given in the Table 6.1 and Table 6.2.
### Control Blocks on Node1

<table>
<thead>
<tr>
<th>Control Block</th>
<th>Name of the Control Block</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ACB - Application Control Block</strong></td>
<td>OIT_CTRL.CS2NODE</td>
<td>Current location of the executable file for the I/O Control Module.</td>
</tr>
<tr>
<td><strong>CCB - Communications Control Block (Local)</strong></td>
<td>NODE1.CS2AP000</td>
<td></td>
</tr>
<tr>
<td><strong>CCB - Communications Control Block (Remote)</strong></td>
<td>NODE2.CS2AP000</td>
<td></td>
</tr>
<tr>
<td><strong>CACB - Communications Access Control Block</strong></td>
<td>CS2AP000.CS2NODE</td>
<td></td>
</tr>
</tbody>
</table>
### Table 6.2 Control Blocks on Node2

**Control Blocks on Node2**

**ACB - Application Control Block**
Name of the control block: DEMO.CS2NODE
Location: Current location of the executable file for the demonstration program.

**ANCB - Alias Name Control Block**
Name of the control block: OIT_CONTROL
Physical Resource: OIT_CTRL.NODE1

**RCB - Remote Control Block**
Name of the control block: OIT_CTRL.NODE1

**CCB - Communications Control Block (Local)**
Name of the control block: NODE2.CS2AP000

**CCB - Communications Control Block (Remote)**
Name of the control block: NODE1.CS2AP000

**CACB - Communications Access Control Block**
Name of the control block: CS2AP000.CS2NODE