

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Low Cost Shop Floor DNC System

**A dissertation presented
in Partial fulfilment of the requirements
for the postgraduate Masters of Technology
in Automation & Control at
Massey University**

William H.Y. Ma

2002

**LOW COST SHOP FLOOR
DNC SYSTEM**

William H.Y. Ma

2002

ABSTRACT

Direct/Distributed Numerical Control (DNC) has a vital role in delivering a successful Computer Integrated Manufacturing (CIM) strategy. DNC is the most popular form of factory automation system in the shop floor environment. Its core function is to enable manufacturing information to flow smoothly and efficiently to and from the shop floor facilities. The current New Zealand small to medium manufacturers are unwilling to make large financial investment in the more expensive packages, and hence, there is a need for a cost effective DNC application software within this sector of the industry.

The research conducted for this project focuses on the application of a multiport serial card, and the development of a low cost DNC application software that can be implemented in the small to medium size companies for transferring data and other manufacturing data such as drawing files, and computerised numerical control (CNC) programs. In addition, the research also looks at methods to allow remote access to the system through the World Wide Web (WWW).

In order to achieve the objectives mentioned above, a powerful and user-friendly user interface programming tool kit — Borland's Delphi 4 was adopted as the key development tool. Delphi 4 is a Rapid Application Development (RAD) package that is fully compatible to the Multiport's serial programming library, and majority of the Microsoft's remote access technology such as Object Linking and Embedding technology (OLE) or ActiveX.

Acknowledgements

I would especially like to thank the following for their time and support throughout this year:

The Lord Almighty for his sovereign support and guidance.

My project supervisor Dr Liqiong Tang, for her invaluable advice and guidance through out the entire year.

My family and friends, who gave me indefinite amount of emotional support.

List of Figures:

Figure 1.1: Proposed interface connections	8
Figure 2.1: AWF's CIM Model	12
Figure 2.2: Application of PC in the VM	16
Figure 2.3: VM Scope & Integration with Enterprise Functions	18
Figure 2.4: DNC Model	23
Figure 2.5: Auxiliary & basic functions in DNC system	24
Figure 2.6: 25 Pin and 9 pin serial connectors	28
Figure 2.7 Asynchronous Serial Data Frame (8E1)	31
Figure 2.8: Traditional modularity concepts	37
Figure 2.9: OO Approach	38
Figure 2.10: Delphi Interface	40
Figure 3.1: Moxa multiport card	43
Figure 3.2: Putting the Multiport Card into the the PCI Slot	43
Figure 3.3: IO-IRQ utility	44
Figure 3.4: Driver installation	45
Figure 3.5: Configuration Panel	46
Figure 3.6: Property Dialog	46
Figure 3.7: Diagnostic software	47
Figure 3.8: Terminal Emulator	48
Figure 3.9: Setting dialog	49
Figure 3.10: Transparent Mode	50
Figure 3.11 Bridge Mode	50

Figure 3.12: Data Scope	51
Figure 3.13: 9 Pin gender changer	52
Figure 3.14: Null modem cable	53
Figure 3.15: The entire cable and adaptors used	53
Figure 3.16: Hierarchical PComm library	54
Figure 4.1: Waterfall model	56
Figure 4.2: Storyboarding	59
Figure 4.4: Final design of main interface	60
Figure 5.1: Delphi's project manager	61
Figure 5.2: Showstatus	63
Figure 5.3: Config.pas	64
Figure 5.4: Output state & flow control	64
Figure 5.5: Confirmation Dialog	65
Figure 5.6: Default name	66
Figure 5.7: Menu Designer	67
Figure 5.8: Anatomy of keyword Function	67
Figure 5.9: Test	70
Figure 6.1: Character Transfer Interface	71
Figure 6.2: Quick button for Character exchange interface	72
Figure 7.1: Confirmation Dialog	78
Figure 7.2: Protocol Specification Tab Sheet	79
Figure 7.3 Directory List box	80
Figure 7.4: Feedback Tab Sheet	83

Figure 7.5: Quick File Transfer Interface	88
Figure 7.7: Machine Form	89
Figure 7.8 Property form	90
Figure 8.1: Changing the FormStyle	92
Figure 8.2: Multiple NC editor	93
Figure 8.3: NC program in editing	93
Figure 8.4: Cut & Copy button enabled	94
Figure 9.1: RegEdit.exe	100
Figure 9.2: Ole Container	101
Figure 9.3: OLE Container in FtransForm	101
Figure 10.1: Import ActiveX Control	104
Figure 10.2: ActiveForm Wizard	105
Figure 10.3: Type Library	106
Figure 10.4 Build the ActiveForm in Delphi	107
Figure 10.5: Web Deployment Options	108
Figure 10.6: ActiveForm on the WWW	111

Table of Contents

Abstract.....	i
Acknowledgement.....	ii
List of Figures.....	iii
Table of Contents	i
1. Introduction	5
1.1. The Research Topic	5
1.2. The Scope of Research.....	6
1.2.1. Interface Card Selection & Integration	6
1.2.2. Software Development	7
1.3. Organisation of Dissertation.....	9
2. Literature Review	11
2.1. Introduction	11
2.2. Computer Integrated Manufacturing (CIM).....	11
2.2.1. Background.....	11
2.2.2. Production Planning and Control (PPC).....	13
2.2.3. Computer Aided Process Planning (CAPP)	13
2.2.4. Computer Aided Quality Assurance (CAQ).....	14
2.2.5. Computer Aided Design (CAD)	14
2.2.6. Computer Aided Manufacturing (CAM).....	15
2.3. Virtual Manufacturing (VM).....	16
2.3.1. Background.....	16
2.3.2. Application of VM	19
2.3.3. Virtual Manufacturing over the Internet.....	21
2.4. DNC System.....	22
2.4.1. Background.....	22
2.4.2. Function of DNC System	22
2.4.3. DNC Applications.....	26
2.5. Manufacturing Communication.....	27
2.5.1. Introduction.....	27
2.5.2. History	27
2.5.3. RS232: The Physical Interface	27
2.5.4. Serial Port	29
2.5.5. Serial Transmission Methods.....	30
2.5.6. Bit Rates	32
2.5.7. UARTS	32
2.5.8. Interrupts.....	34
2.5.9. IO Address & IRQ	35
2.6. Software Development Process	36
2.6.1. Introduction.....	36

2.6.2.	Objects & Object-Oriented Software	36
2.6.3.	Borland's Delphi 4.0	39
3.	Multiport Card Installation.....	42
3.1.	Introduction	42
3.2.	Background.....	42
3.3.	Hardware Installation	43
3.3.1.	Quick Hardware Installation.....	43
3.3.2.	Hardware Installation with IO-IRQ Utility	44
3.4.	Software Installation	45
3.4.1.	Installing Driver	45
3.5.	Serial Programming Tools.....	47
3.5.1.	Diagnostic Tool.....	47
3.5.2.	Terminal Emulator	48
3.5.3.	Data Scope.....	49
3.6.	Cabling	52
3.6.1.	9 Pin Female to Female Gender Changer.....	52
3.6.2.	The Null Modem Cable	52
3.6.3.	Installing the Cable	53
3.7.	PComm Application Programming Interface (API)	54
4.	DNC Software Development Process.....	55
4.1.	Introduction	55
4.2.	Requirement Analysis & Definition	56
4.2.1.	End- User/Operating Environment Analysis	56
4.3.	Interface Design.....	58
4.3.1.	Specification Defined	58
4.3.2.	Design Concepts	59
4.3.3.	Final Design.....	60
5.	Establishing Communication	61
5.1.	Introduction	61
5.2.	Establishing Communication.....	61
5.2.1.	PComm.pas	62
5.2.2.	MxTool.pas	62
5.2.3.	ExGlobal.pas.....	62
5.2.4.	Config.pas.....	64
5.2.5.	FormCreate Procedure.....	65
5.2.6.	Menu.....	66
5.2.7.	PortSet Function.....	67
5.2.8.	Open Port Function	68
5.2.9.	Setting1Click Procedure.....	69
5.2.10.	PortOpenClick/ PortCloseClick Procedure	69
5.2.11.	Testing for Signals	69

6.	Character Exchange Interface	71
6.1.	Introduction	71
6.2.	SimpleM.pas	71
6.3.	Threading Applications	72
6.3.1.	The Thread Class	73
6.3.2.	ReadThread.pas	74
6.3.3.	Advantage of Single Thread	75
7.	File Transfer	76
7.1.	Introduction	76
7.2.	File Transfer Protocols	76
7.3.	File Transfer Interface	78
7.3.1.	Protocol Configuration Tab	79
7.3.2.	Directory List Tab	80
7.3.3.	Transfer Status Feedback Tab	83
7.3.4.	FtProc Thread	83
7.3.5.	Execute ()	84
7.3.6.	xCallBack/rCallBack ()	85
7.3.7.	ProcessRet()	86
7.4.	CNC Machine Quick Access	87
8.	MDI Programming	91
8.1.	Introduction	91
8.2.	MDI- Main Window Form	91
8.3.	NC Code Editor	92
8.4.	Child.pas	94
8.4.1.	FileNewClick Procedure	95
8.4.2.	FileOpenClick Procedure	95
8.4.3.	SaveAs Procedure	95
8.4.4.	FileSaveClick Procedure	95
8.4.5.	FilePrintSetupClick Procedure	96
8.4.6.	FilePrintClick Procedure	96
9.	COM & OLE Automation	97
9.1.	Introduction	97
9.2.	Component Object Model (COM)	97
9.2.1.	<i>Interfaces</i>	97
9.2.2.	IUnknown	98
9.2.3.	GUID	98
9.3.	Object Linking Embedding (OLE)	99
9.4.	OLE Object	99
9.5.	MDI OLE Container Interface	100
9.5.1.	ChildWin.pas	102

10.	Remote Access using Active X	103
10.1.	Introduction	103
10.2.	Understanding ActiveX.....	103
10.2.1.	Installing ActiveX Component	104
10.2.2.	ActiveForm Wizard.....	105
10.2.3.	Type Library	105
10.3.	Building the Form.....	107
10.4.	Deploying an ActiveForm.....	108
10.4.1.	Connecting to an ActiveForm.....	110
11.	Testing.....	112
11.1.	Introduction	112
11.2.	Interface Testing	112
11.3.	Test Results.....	113
12.	Results & Discussions.....	114
13.	Recommendations.....	115
14.	Conclusion.....	116
15.	References	117
	APPENDIX A.....	119

CHAPTER 1

1. Introduction

1.1. *The Research Topic*

In the modern industrial production environment, the number of computer application is steadily increasing. The potential for automated process, especially in manufacturing sector, has been put into practice by applying ever more sophisticated computer aided tools and methods.

With the increase saturation of companies with computerised processes. The future development of such beneficial integration will not only lie in the installation of isolated computerised solutions in the different areas (island solutions), but rather in the coupling and therefore in the utilisation the combined effect of automation technology and shopfloor management. Such concept is often referred to as Computer Integrated Manufacturing (CIM), which includes the coupling of all areas linked to the actual manufacturing operations, including Computer Aided Design (CAD), Computer Aided Process Planning (CAPP), Computer Aided Manufacturing (CAM) and Material Requirement Planning (MRP).

As part of the key development in achieving CIM, Direct/Distributed Numerical Control (DNC) and Computer Numerical Control (CNC) were introduced as the first computer control systems in the early 1970 [4]. The DNC control system establishes a direct link between a computer and each NC machine tool, and eliminates the necessity for using punched tape input. In addition, CNC technology consists of a soft-wired controller that can be adapted to various types of machine tools by programming the control functions into the computer memory for a particular machine. Today, the DNC system is a basic self-contained control unit in the manufacturing automation environment, and one of the first steps towards factory automation based for CIM.

DNC is a major necessity for any manufacturing organisation that wants to achieve a successful CIM. However, the costs of employing existing DNC system packages have driven away many small and medium companies within the New Zealand industry. Hence, by developing a low cost shop floor DNC system will be very appealing for this market sector. This research project, led by Dr Liqiong Tang, Institute of Technology & Engineering (ITE) of Massey University, is dedicated to the development of low cost DNC systems that will allow further in depth integration of CIM for the New Zealand manufacturing industry.

1.2. The Scope of Research

The research undertaken here concerns the critical information flow between different users and seeks a low cost and efficient method in delivering manufacturing information between the geometrical/technological domains, this domain area usually only includes the CAD/CAM/CNC Data Communication.

The system developed in this Masters thesis involves the implementation of a suitable interface card and the development of Graphical User Interface (GUI) that will integrate with existing CAD/CAM application in ITE faculty.

1.2.1. Interface Card Selection & Integration

With the aim of connecting the isolated CNC machines, to the current industrial communication systems. The DNC system developed within this research project will implement several serial communication hardware devices.

One of the keys of the research is to understand the hardware system required to deliver a cost effective Serial Distributed Control System. Therefore, factors such as the following will be considered during the selection of the interface card

- Highest performance that meets all speed-demanding and data intensive communication needs.
- Large on-board buffer for high-performance communication
- Compact design size—ideal for high performance systems
- Critical industrial control
- Response demanding monitoring systems
- Cost Efficient

1.2.2. Software Development

A large number of small and medium size companies today possess a significant number of machines and equipment incompatible with the new Operating System (OS) standards. For control of data transfer, there are still many DOS application that can only run single task at a time, which means no other application program can be executed concurrently. Another disadvantage of these DOS application is the lack of communication between application programs such as CAD/CAM, which results in large amount of time wasted in switching between programs and loading saved data.

Therefore, the application program developed in this project will focus on the following core functionality to overcome the shortcomings within the DOS environment.

- Front-end Application interface for CAD/CAM Environment
- Communication interface programs for both serial data transfer and networking between the CAD/CAM workstations and CNC machines.
- Application programs for NC code editing and for accessing other software application within the workstation.
- Remote software access through Internet for cost effective software distribution.

With the previous listed functions, Figure 1.1 illustrates three interfaces that are considered within this research.

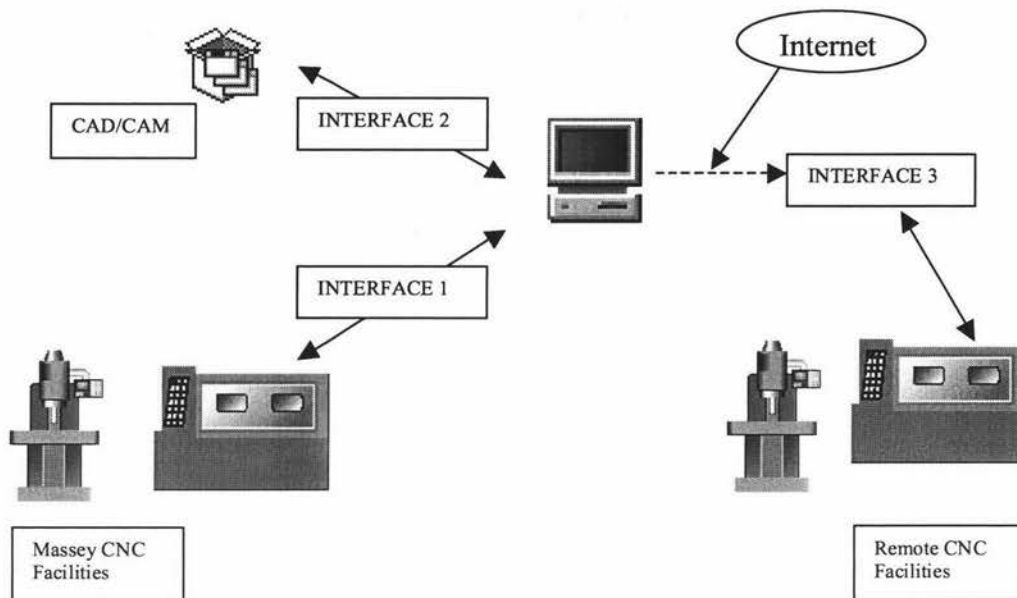


Figure 1.1: Proposed interface connections

- *Interface 1* --- For communication with CNC machines
- *Interface 2* --- For communication with CAD/CAM System and other desktop applications
- *Interface 3* --- Internet interface for remote access to software from remote site.

The front-end application interface within this research project is designed through Borland's Delphi 4. Delphi offers all of the windows GUI advantages, which allows the programmer to write standard window features such as title bar, menu bar and common tool bars for quick access to commonly used commands.

1.3. Organisation of Dissertation

The dissertation is arranged as follows:

Chapter 2 deals with the methods and theories of current shopfloor manufacturing strategies, concepts such as Computer Integrated Manufacturing (CIM), Virtual Manufacturing, Direct/Distributed Numerical Control (DNC) and serial communication devices are described in detail. This chapter aims to provide essential theoretical support and research direction for this project.

Chapter 3 focuses on the installation process of the selected multiport card, and describes some of the serial programming utilities that accompany the multiport card.

Chapter 4 covers the software engineering process applied. The chapter only covers the analysis and design stages. Allowing the implementation and testing stages to be described in later chapters.

Chapter 5 describes the basic steps used for establishing communication interface to the multiport device. The chapter denotes some of the fundamental functions used within the program, and illustrates the testing results ascertained through a RS-232 tester.

Chapter 6 illustrates the implementation of the character exchange interface. This section describes the application of Threading function from Delphi, allowing user to view the keyboard input from the other end of the serial network.

Chapter 7 denotes the definition of several serial communication protocols, and explains how these protocols are implemented through the Application Programming Interface (API) functions

Chapter 8 describes the methods used in implementing Multiple Document Interface (MDI) function within this project. The MDI interface will be used for the NC editor and OLE child form. This chapter will describe the code used in constructing the NC editor.

Chapter 9 discusses the application of Component Object Model (COM), and Object Linking & Embedding (OLE) technology within the research thesis. The chapter will also describe the code used constructing the OLE child form

Chapter 10 focuses on the use of ActiveX technology, describing how the resulting interface will allow remote access to the restricted version of the interface through the World Wide Web (WWW).

Chapter 11 describes the testing procedures conducted on the interface program.

Chapter 12-14 discusses the results of the project by listing areas of work done and possible areas of improvement.

CHAPTER 2

2. Literature Review

2.1. Introduction

This chapter reviews the theoretical background required for developing a low cost shop floor Distributed Numerical Control (DNC) system. Therefore, it aims to cover areas within the study of factory automation, and the theoretical knowledge required.

2.2. Computer Integrated Manufacturing (CIM)

2.2.1. Background

The concept of Computer Integrated Manufacturing (CIM) is based on the principle of the production cycle developed by Harrington. Harrington's definition of the production cycle starts with the development of a product, then moves on to manufacturing and delivery to the customer and ends with service and maintenance. These functions are closely interconnected within the operational sequence organisation, it is not expedient to regard the single functions as isolated from each other. Hence by extending Harrington's concept, the components involved in CIM are the data processing tools underlying the production cycle.

Although the concept of CIM has been around more than 2 decades. Much of its definition differs widely, due to the fact that much has been written with regard to the substance of CIM without a comprehensive consensus.

In order to provide a clarified definition of the components attributed to CIM within the research project. It is intended to do so by referring to the publication of *Ausschuss fuer Wirtschaftliche Fertigung e.V* (AWF-Committee for Economical Manufacturing). Waldner's literature [3] describes AWF in the sense that all data processing system in every area related to manufacturing, which are to be integrated.

The term CIM described by AWF denotes the use of computer to design the products, plan the production, control the operations, and accomplish many of the business related function in a manufacturing firm. The theory also suggests convergence of various functions within manufacturing by means of computer systems. It is in the data processing and information flow within a firm that this integration primary occurs and the control of production equipment by computers will facilitate integration in a CIM system.

Figure 2.1 illustrates the graphical representation of the system. Its aim is to achieve an integration of the technical and the administrative functions involved in manufacturing.

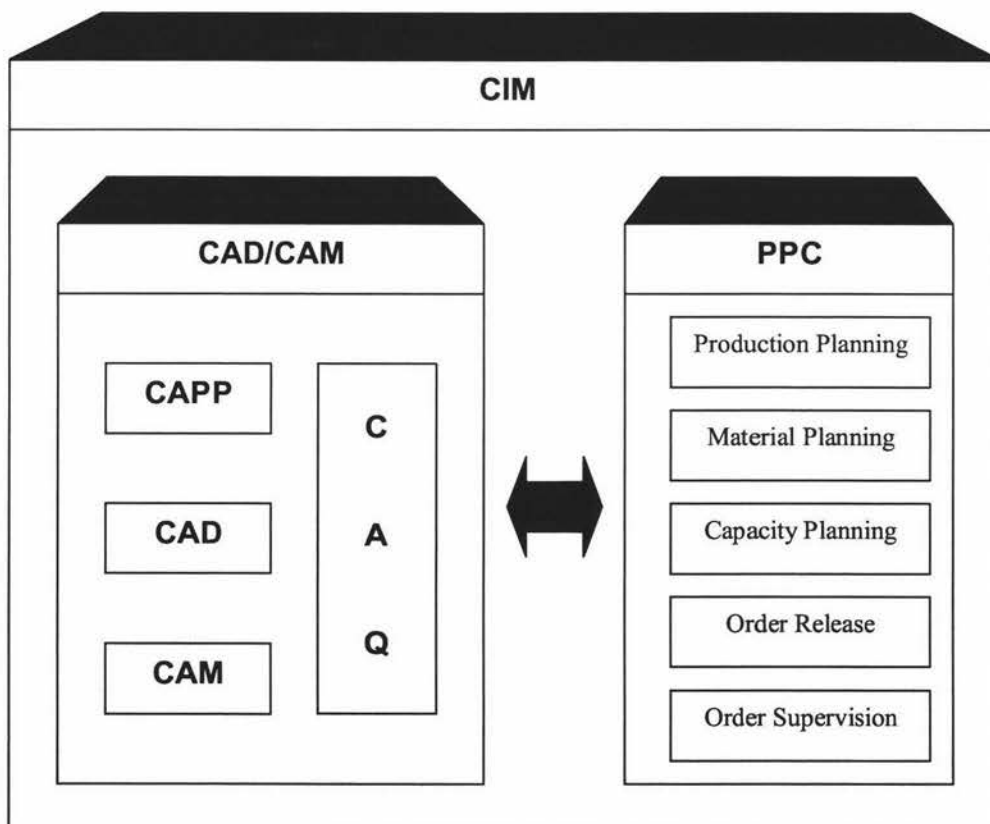


Figure 2.1: AWF's CIM Model [3]

The figure is introduced to represent an overall unified computer system, which allows the factory to achieve manufacturing productivity. Based on the definition given by Dorf [7], it states a full CIM system would provide a centralised control of the manufacturing environment, addressing two main requirements: vertical integration, and horizontal integration.

The term vertical integration describes Computer Aided Design (CAD), Computer Aided Process Planning (CAPP), Computer Aided Manufacturing, and Computer Aided Quality Assurance (CAQ) that is illustrated within the left hand side of Figure 2.1. It describes the capability of the computer system to integrate the full process from design conception to part manufacture.

The term horizontal integration describes a hardware/software network solution, encompassing all the functionalities exercised on the manufacturing floor. Such a capability would include the Production Planning and Control Systems (PPC).

2.2.2. Production Planning and Control (PPC)

Production Planning and Control (PPC) Systems are applied as a higher-level instrument for the organisational planning, control and supervision of the production processes with regards to volume, delivery dates and capacities [7].

2.2.3. Computer Aided Process Planning (CAPP)

CAPP Systems are applied in the area of computer aided process planning. It uses the results generated in the design process. CAPP system cover data processing systems, which support the planning of the operation process and the operation process sequence as well as the selection of methods and resources necessary for the manufacturing of the objects and the control program for resources [7].

2.2.4. Computer Aided Quality Assurance (CAQ)

Quality assurance is included as a part of the pre-production functions but it also overlaps into the areas of production. CAQ indicates the support by data processing tools of the planning and execution of quality assurance, which on the one hand includes the generation of test schemes, test programs and control values and on the other hand the execution of measuring and test methods [7]

2.2.5. Computer Aided Design (CAD)

Computer Aided Design (CAD) system within the CIM system aims at the engineering design function, rather than the traditional drafting function. The distinction is based on the paradigm where the drafting function provides support for the design function. Another area that sets the computer-aided design is the incorporation of a more extensive set of analytical tools. This is an additional function responds to the needs of both the design and drafting functions.

Furthermore, such systems tend towards the full integration of the entire Design-To-Manufacture (DTM) process. A trend that is driven by concepts as the ones listed below

The Paperless Factory

Traditionally, the engineering and drafting functions are usually carried out by separate groups within the engineering department, where engineering designs are passed to drafting departments for the detailing process and the production of engineering drawings. However, one aspect of introducing the computer aided design technology is the move towards paperless industry, which transform the DTM process to depend more on "computerised-databases" and less on engineering (paper) drawings. Furthermore, the concept of paperless factory extends all the way to the manufacturing functions, and with assistance from some other concepts it facilitates the full integration of DTM process [7].

1. *Feature-Based Design*

The concept of feature-based design was advanced as an attempt to simplify mechanical process design as an extension of CAD. As described by Dorf [7], technical drawing can not supply sufficient information for all kinds of application programmes in a CAD/CAM system. However the introduction of feature-based solid modelling is expected to resolve this problem. By using the feature-based modelling techniques, a designer can identify the geometry of a part that corresponds to a particular machining operation. Feature based modeller use machining terms such as "bore" to make a hole, rather than the Boolean operation found in traditional packages. Commercially available feature-based modeller such as SolidWorks, and Pro Engineer provides capabilities to create a mechanical design consist of a collection of standard geometric features, accompanied by standard processes to create these features. The list of such features includes prismatic shapes, cylinder, cones, slots, holes, rounds, fillets, etc.

2.2.6. Computer Aided Manufacturing (CAM)

The application of CAM systems in CIM relates to the areas of production executing functions. The system could normally consist of single CNC machines, several numerically controlled manufacturing installations that are controlled by a host computer through a DNC system. Thus, providing the ability to form Flexible Manufacturing Cells (FMC) around the machines within any work shop [7].

2.3. Virtual Manufacturing (VM)

2.3.1. Background

The significant growth in the computer technology sector has also influenced and improved the concept of CIM over the past decade. One of the core developments evolved from CIM was the concept of Virtual Manufacturing (VM). Figure 2.2 illustrate the vision of VM is to provide a capability to “Manufacturing in the Computer”. In Lin’s literature on VM [16], it describes that VM will provide a modelling and simulation environment so powerful that the fabrication/assembly of any product, including the associated manufacturing process, can be simulated in the computer. This powerful capability essentially takes all of the variables in the production environment from shop floor processes to enterprise transaction. In essence, VM accommodates the visualisation of the process in CIM, and has even greater impact on related process such as accounting, purchasing and management.

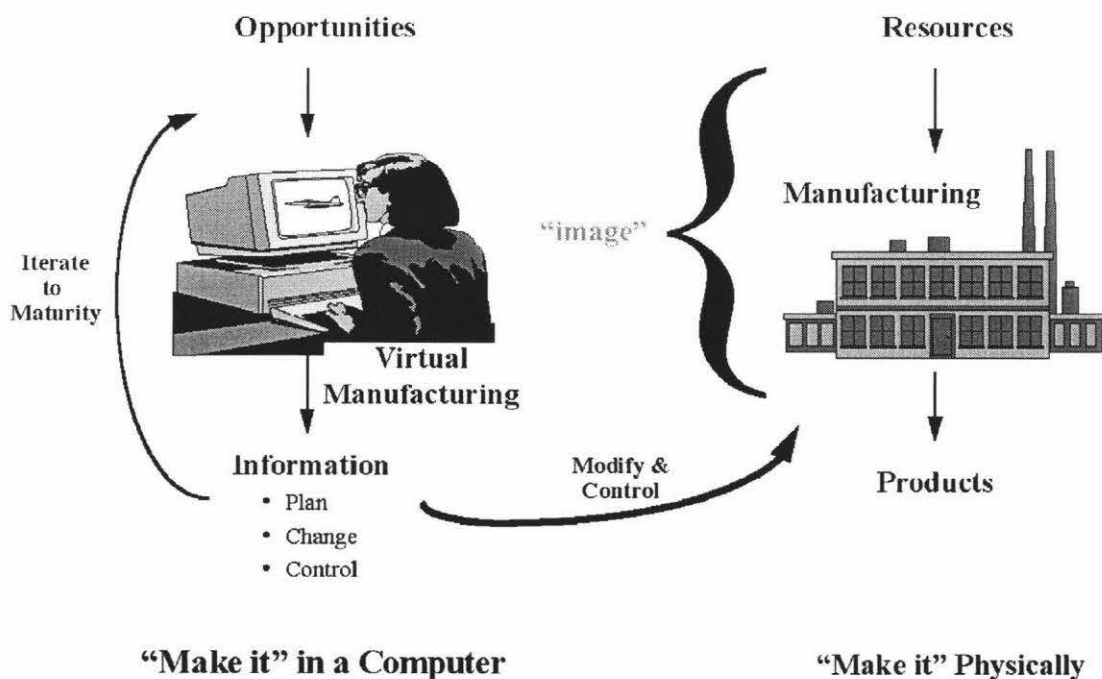


Figure 2.2: Application of PC in the VM [16]

According to research conducted by Lin [16]. Two major research events have combined to initiate the concept of VM. First the on going improvement on US defence environment and the acquisition strategies required development of the capability to prove the manufacturability and affordability of new weapons system prior to the commitment of large production resources. VM has the potential to address these issues. Secondly, through out the last decade the engineering/manufacturing sector has made tremendous advancement in modelling and simulation technologies. Thus, offering a realistic opportunity to build such a computing capability [16]. For example, the Distributed Interactive Simulation (DIS) program has demonstrated the usefulness of Modelling and Simulation (M&S) in an environment rivalling manufacturing in complexity.

Developing a definition of something as complex as VM is often difficult; such a definition can rarely capture everything necessary to fully capture the complexity. As a result, selected commentary is presented below to better capture some of its complexity.

Perhaps the closest definition for VM is [16]:

- *VM is an integrated, synthetic manufacturing environment exercised to enhance all levels of decision and control. It focuses on improving manufacturing processes by the employment of a model-based approach which leverages simulation capabilities*

The fundamental notion of VM is that it is a computer-based, simulated product development environment that enables the manufacturers to "make it virtually" before "making it for real". The term "product development" encompasses all of the various activities, both business and technical, associated with developing and producing a given product.

Figure 2.3 represents an attempt to capture the idea that while VM is not just a new “buzz word” that is used to accomplish the desired cross-functional trade-off analyses; in most cases it can be integrated with all of the relevant enterprise functional areas via a trade-off mechanism, the Integrated Product and Process Design (IPPD process).

VM must be scoped if we are to achieve meaningful near-term

Suggestion:

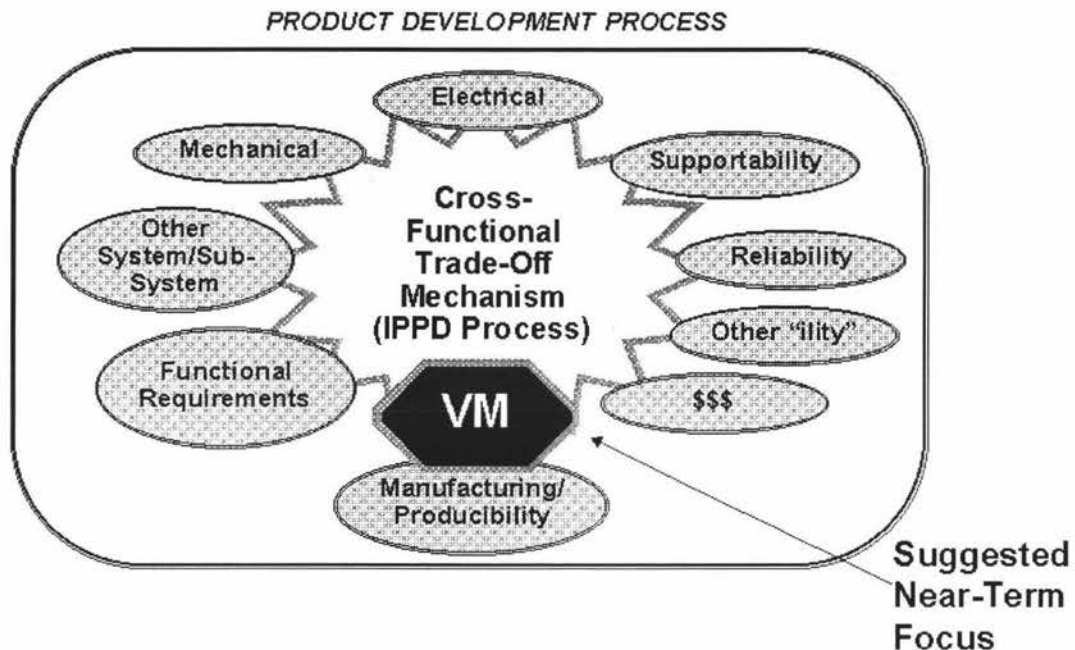


Figure 2.3: VM Scope & Integration with Enterprise Functions

VM allows for the creation of many more "soft prototypes" (by reducing both cost and time factors), and/or reduces the cost of the prototyping process overall.

VM is model-based manufacturing, with tools that leverage those models. Primary among the techniques used is simulation, which can reduce some costs of manufacturing and allow exploration of many options in a mixed real/computed space.

At the local level, VM adds simulation to control processes to allow for expedited re-engineering/improvement of processes. At a more global level, VM provides for evaluation of partial and complete designs by "manufacturing in computers" in an enhanced IPPD environment. VM is not a single solution, architecture or monolithic database approach. It is a collection of many smaller, incrementally tools, together with some more overarching concepts that may require larger investments by developers and users.

2.3.2. Application of VM

The following categorization shows the breadth of areas in which VM might be used [16].

CORPORATE MEMORY -- Through the increased development and use of expert systems to capture the knowledge of subject matter. The first area which the VM will enhance is the corporate memory. The details of product design are captured as part of the corporate memory in a systematic way, but the manufacturing process details often are not. Using expert systems in conjunction with VM would be a significant improvement by providing process capability and cost information to guide the product design process as well as adding some viability to the concept of "shelf technology" where a product might go into production long after the initial design prototyping and testing are completed [16].

SUPPLIER MANAGEMENT – The current VM impact on suppliers is rare and the use of VM by suppliers themselves would often be limited to the larger companies because of the anticipated large investment required to install VM [16]. The future impact on supplier management, however, is expected to be very significant. Make/buy decisions will be enhanced through easy access to better quality and more detailed information on costs, capacity, process capability and lead-times as part of the make/buy decision process.

Cost control would also be enhanced because VM offers more accurate cost information for suppliers. Major suppliers will have early involvement in product design and process planning through the Integrated Product and Process Design (IPPD) teaming approach that is likely to be an accelerating and long-lasting trend and will interact with VM in that context. Smaller suppliers are also likely to see some positive impacts by getting much better and more stable product requirements information from customers and the customers should be positively impacted by not having to invest so many resources in having to solve problems with their suppliers [16].

PRODUCT DESIGN -- The emerging modelling and simulation will enhance the effectiveness of systems integration in the design process. It will allow organisations to minimize interference between subsystems and, and reduce the dependence on hard-mock-ups. Also in the near term, electronic co-location of IPPD team members will become more practical and widespread. In the longer term, major improvements to the transition from design to production are envisioned because of much stronger and more effective influence of process capacity and manufacturing cost information on the product designer as well as the ability to do many more design iterations prior to committing to hardware.

COST ESTIMATING -- The move towards VM will provide more accurate cost information than can typically be provided by current cost accounting systems. This will accelerate the current trend toward activity-based accounting systems and other accounting system changes that allow detailed and accurate product costing. Some current reliance on "semi-expert" systems for cost estimating was identified, VM provides better data through more accurate approaches. In near future, VM systems will provide accurate cost data throughout the design, development, and production process. Cost estimating systems will become fully integrated with design and manufacturing databases and will have access to detailed process-level design feature related data [16].

SHOP FLOOR – The application of VM will allow shop floor workers to have a greater influence on the design process, and manufacturing approaches that have been modelled and simulated above the shop floor will be brought out on the shop floor to validate the models and simulations. Significant improvements to work instructions will be seen through the ready availability of graphics. Much better tooling will be available on the shop floor with features that make it easier for the worker to succeed via access to better instructions and illustrations to promote error-free tool use. This will also make it easier to accommodate the envisioned drop in the average skill and education level of shop-floor workers. The proofing of designs and manufacturing processes in the computer prior to commitment to hardware should sharply reduce the problems on the shop floor.

2.3.3. Virtual Manufacturing over the Internet

The Internet has recently become a major information resource provider for industry, and its demand keeps growing. The World Wide Web (WWW) plays an especially important role in providing information services on the Internet.

Several languages are currently used or developed in the near term, there are promising potential areas for development of VM tools on the internet. The best known example is the HTML language used for constructing the documents read by web browsers. Documents written in HTML include embedded commands that change the formatting of the text or specify remote locations from which further information can or should be retrieved.

HTML documents may be useful for sending certain kinds of Virtual manufacturing information over the Internet, by transmitting design and manufacturing data to a web browser from a program running at a remote location. Example includes, CAD models, and manufacturing data such as NC programmes. Other application of Internet includes software product demos, where GUI can be developed through advanced Client/Server technology such as ActiveX to distribute the software over the Internet.

2.4. DNC System

2.4.1. Background

The New Zealand manufacturing industry mainly employees numerically controlled machine tools because it provides a fast and low-cost adaptation to the changing production requirements for the small quantities of products.

However, when machines have to be frequently re-set, coupled with short production runs, the time spent in delays due to setting and loading of NC programs at different terminals has an undesirable effect on the efficiency of machine utilization.

DNC (Direct or Distributed Numerical Control) refers to controlling machines or machine cells (a group of related machines) using a centralised computer. DNC system originally had machines without individual controllers that were all controlled from a central computer (Direct Numerical Control) [8]. However, with the continue increase in the number of NC machine within the factory floor. The current form of DNC is typically used to describe a system in which each machine has its own controller that is linked to a central computer system that co-ordinates the machines and maintains a list of part programs that can be downloaded to a machine as needed; this form of DNC is more accurately know as Distributed Numerical Control.

2.4.2. Function of DNC System

The basic function of a fundamental DNC system is to download and upload NC programs -- character by character, or block-by-block or even whole program, depending on the target CNC controller's memory capacity. It is very convenient and flexible compared with paper tape. However, as described in the previous sections, there are more functions that can be explored from a DNC system in relation to the CAM sector of the CIM strategy.

Smith [4] describes a complete DNC control system should consist of the following four basic components

1. DNC Computer
2. Bulk Memory used as a local database for NC part programs and other manufacturing information
3. Telecommunication lines and relevant adaptors
4. Machine tools & controllers

Figure 2.4 illustrates a DNC model, which includes four components as mentioned above. The DNC computer downloads the part program from disk to a specific machine as the need arises. It also receives data back from machines.

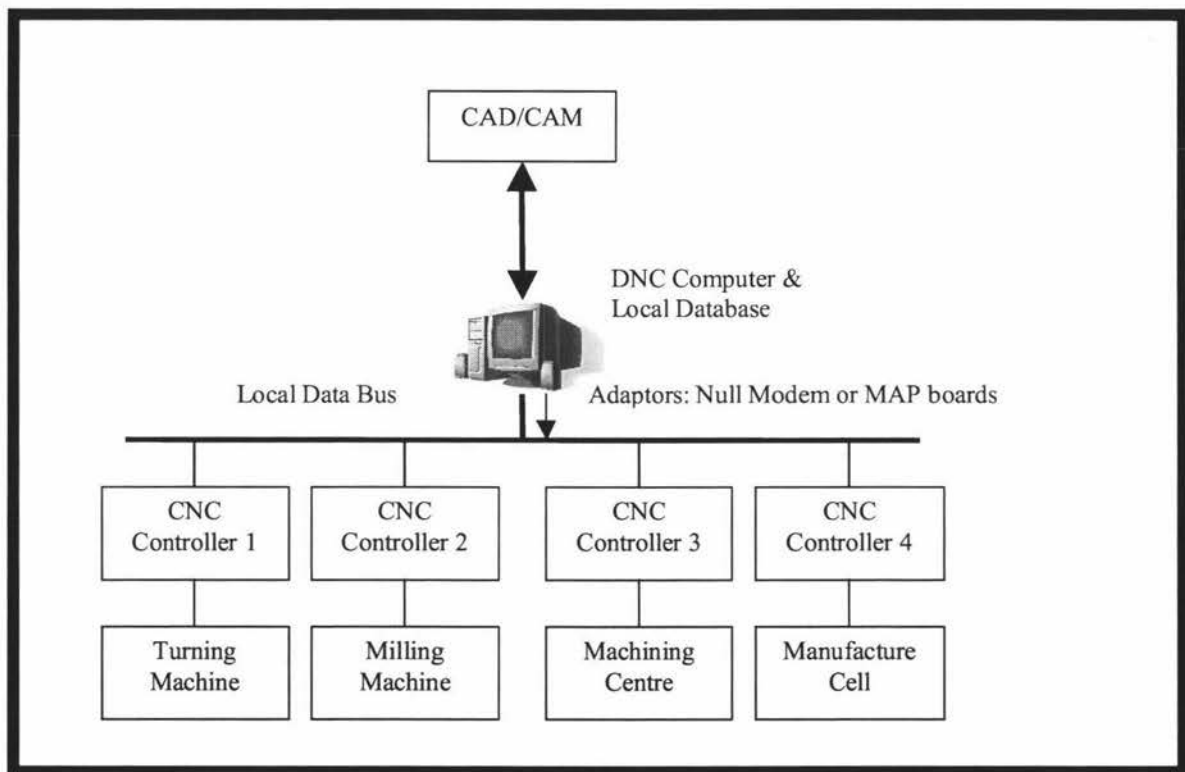


Figure 2.4: DNC Model [4]

In a DNC system, Figure 2.5 shows that there are 2 kinds of functions, i.e. basic functions and auxiliary functions, may be identified, irrespective of the type of control to which it is connected.

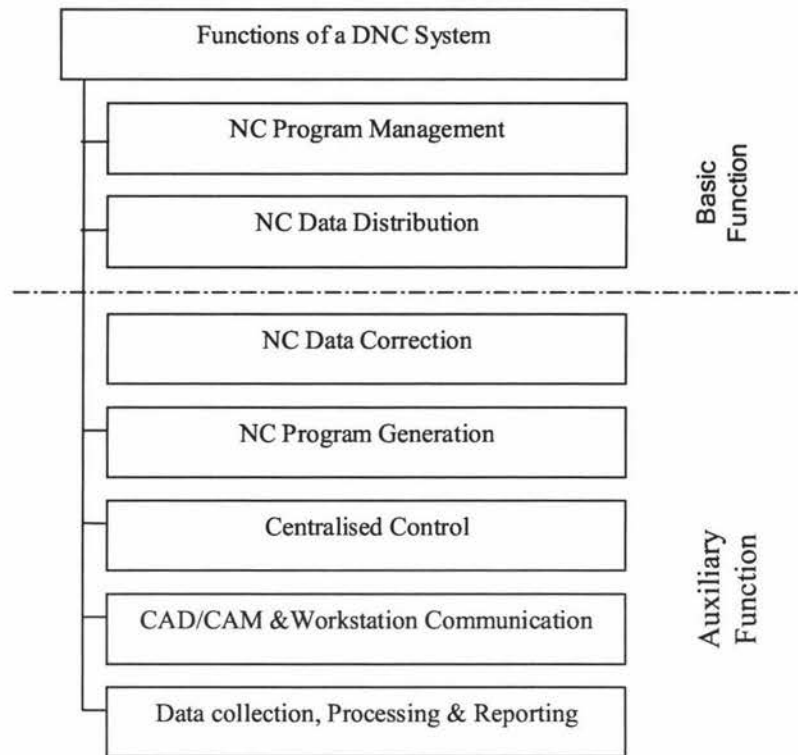


Figure 2.5: Auxiliary & basic functions in DNC system

2.4.2.1. Basic Functions

The basic function covers the management of the management of the NC part program and the distribution of the NC data at the appropriate time to the respective control unit via the production computer. The management of large volumes of NC data in a production computer offers considerably more scope for storing and working with NC programs than the traditional punched tapes. Once the program is generated, the data are read in to the external memory store of the production computer so that they may be managed or administered, which include copying, recording, suppressing and releasing [4].

The NC data distribution comprise making the NC programs available and transmitting the NC data to the machines. When the machine operator requests a particular NC program the production computer will firstly test the compatibility and look for any barring with respect to the NC unit and the NC program.

The data output buffer is prepared and filled an accept or reject message is transmitted to the requesting terminal in accordance with the results of the tests. If this is an acceptance signal the transmission (in blocks, words or individual characters) is dependent upon the hardware configuration of the complete system. The refilling of the buffer in the central memory stores from the external memory store occurs automatically through the software system in the production computer (DNC program system) [4]. After an NC program has been completely transmitted, the process is closed and the particular channel which was used is made available for other requirements.

2.4.2.2. Extended Functions

In order that the NC part programs are carried out effectively and so that any necessary corrections may be included the NC data correction in the computer should be effected through the operator's terminal directly on the machine itself. The computer aided generation of the NC code will provide the functionalities needed to form VM by allowing simulation and testing to be done prior to manufacturing.

Beside serial communication for NC code transmission to CNC machines, data communication over the network is required to accomplish system integration. Communication among the various systems is such an important function that is central to the operation of any DNC system.

The essential communications are done through a centralised workstation, providing links and communication between the following components of the system:

1. DNC Workstation & Machine Tools
2. DNC Workstation and the CAD/CAM application or other Workstation

The DNC system is also likely to have some form of programs such as a monitoring program and a shop floor control program to enhance its capability. The basic purpose behind the data collection, processing and reporting is to monitor production in the factory. Data are collected on production piece count, tool usage, machine utilisation, and other factors that measure performance in the shop floor. These data must then be processed by the DNC computer and reports any critical trends to management.

2.4.3. DNC Applications

Key features need to be considered when evaluating a DNC application:

ERROR CHECKING FOR DATA COMMUNICATION — Error checking of data communication is very important in real time manufacturing on the shop floor. No error checking programs will result in possible heavy loss of data caused by the high level of interference that subsides within the industrial environment

SYSTEM SECURITY — Depending upon the real manufacturing environment, it is important to maintain levels of control over system use and access. Inability to manage the possibility of misuse could often lead to poor product quality, since it is impossible to track the user that caused the defect.

USER FRIENDLY MENU & STRUCTURE — The user interface for DNC operating system should be user friendly. It is important to realise that most operators within the shop floor are not required to have detail knowledge in software application. Therefore a well designed interface will, in some extent, decrease the skills demanded from an operator.

2.5. Manufacturing Communication

2.5.1. Introduction

One of the approaches in improving the manufacturing efficiency is to allow the user to access manufacturing data quickly and efficiently. To do so, the designer of such system must have a clear understanding of the industrial communication equipments behind most of the CNC machines. Therefore it is the objective of this section to cover detailed studies conducted on Serial communication and the components used.

2.5.2. History

RS-232 was originally adopted in 1960 by the Electronic Industries Association (EIA). The standard evolved over the years and in 1969 the third revision (RS-232C) was to be the standard of choice of PC makers. In 1987 a fourth revision was adopted (RS-232D also known as EIA-232D). In most part of this new revision, 3 additional test lines were added. In this section several parts of the original RS-232C standard and mostly the ones used in the PC world [13].

2.5.3. RS232: The Physical Interface

This section denotes the basic understanding of RS-232 connection to allow work to be carried out with cables, connector, cards, and wires.

Most equipment using RS-232 serial ports use a DB-25 type connector that is illustrated in Figure 2.6. Based on the literature research conducted [17], many PCs today use DB-9 connectors (see Figure 2.6) since all it is required in asynchronous mode is 9 signals. Normally the male connector is on the DTE (Data Terminal Equipment) side and the female connector is on the DCE (Data Communication Equipment) side even if this is not always the case.



Figure 2.6: 25 Pin and 9 pin serial connectors

The standard specifies 25 signal pins, and that the DTE connector should be a male and the DCE connector should be a female. The most used connectors are the DB-25 male, but many of the 25 pins are not needed. For that reason in many modern PCs a DB-9 male connector is used. So it is common to find one or more of these connectors in the rear panel of the PC. The voltage levels are between -3V and -15V for a logic high. A logic low is a voltage between +3V and +15V. The commonly used voltages are +12V and -12V. The most commonly used signals are listed below [17]:

25 Pin #	9 Pin #	SIGNAL NAME
20	4	DTR (Data-Terminal-Ready): The PC tells the modem that is powered up and ready to send data.
6	6	DSR (Data-Set-Ready): The modem tells the PC it is powered up and ready to transmit or receive data.
4	7	RTS (Request-To-Send): The PC sets this signal when has a character ready to be sent.
8	1	CD (Carrier-Detect): The modem sets this signal when has detected the computer.
5	8	CTS (Clear-To-Send): The modem is ready to transmit data. The computer will start sending data to the modem.
7	5	This is the logical ground which is used as a point of reference for all signals received or transmitted.
22	9	RI (Ring Indicator) This line is used mostly by communications software when the modem is not in "auto answer" mode.
2	3	TxD: The modem receives data from de PC.
4	2	RxD: The modem transmits data to the PC

Table 2.1: Pins description [17]

2.5.4. Serial Port

The serial port is an I/O (Input/Output) device. An I/O device is a way to get data into and out of a computer. There are many types of current types of I/O devices, such as serial ports, parallel ports, disk drive controllers, Ethernet boards, universal serial buses, etc.

In most cases PCs have a 9-pin connector (sometimes 25-pin) on the back of the computer. Computer programs can send data (bytes) to the transmit pin (output) and receive bytes from the receive pin (input). The other pins are for control purposes and ground.

According to Mark [13], the serial port converts the data from parallel to serial and changes the electrical representation of the data. Inside the computer, data bits flow in parallel (using many wires at the same time). Serial flow is a stream of bits over a single wire (such as on the transmit or receive pin of the serial connector). For the serial port to create such a flow, it must convert data from parallel (inside the computer) to serial on the transmit pin (and conversely). Data is transferred from sender to receiver one bit at a time through a single line or circuit. The serial port takes 8, 16 or 32 parallel bits from the computer bus and converts it as an 8, 16 or 32 bit serial stream. The name serial communications comes from this fact; each bit of information is transferred in series from one location to another [13].

In theory a serial link would only need two wires, a signal line and a ground, to move the serial signal from one location to another. But in practice this doesn't really work for a long time, some bits might get lost in the signal and thus altering the ending result. If one bit is missing at the receiving end, all succeeding bits are shifted resulting in incorrect data when converted back to a parallel signal. So to establish reliable serial communications the PC must overcome these bit errors that can emerge in many different forms. Bits from the computer bus and convert it as an 8, 16 or 32 bit serial stream.

2.5.5. Serial Transmission Methods

Two serial transmission methods are used that correct serial bit errors. The first one is synchronous communication, the sending and receiving ends of the communication are synchronized using a clock that precisely times the period separating each bit. By checking the clock the receiving end can determine if a bit is missing or if an extra bit (usually electrically induced) has been introduced in the stream [17].

To illustrate this method of communication, let's say that on a conveyor belt a product is passing through a sensing device every 5 seconds, if the sensing device senses something in between the 5 second lap it assumes that whatever is passing is a foreign object of some sort and sounds an alarm, if on the 5 second lap nothing goes by it assumes that the product is missing and sounds an alarm. One important aspect of this method is that if either end of the communication loses its clock signal, the communication is terminated [17].

The alternative method (used in PCs) is to add markers within the bit stream to help track each data bit. By using a start bit which indicates the start of a short data stream, the position of each bit can be determined by timing the bits at regular intervals, by sending start bits in front of each 8 bit streams, the two systems don't have to be synchronized by a clock signal, the only important issue is that both systems must be set at the same port speed. When the receiving end of the communication receives the start bit it starts a short-term timer. By keeping streams short, there's not enough time for the timer to get out of sync. This method is known as asynchronous communication because the sending and receiving end of the communication are not precisely synchronized by the means of a signal line.

Each stream of bits are broke up in 5 to 8 bits called words. Usually in the PC environment you will find 7 or 8 bit words, the first is to accommodate all upper and lower case text characters in ASCII codes (the 127 characters) the latter one is used to exactly correspond to one byte. By convention, the least significant bit of the word is sent first and the most significant bit is sent last. When communicating the sender encodes the each word by adding a start bit in front and 1 or 2 stop bits at the end. Sometimes it will add a parity bit between the last bit of the word and the first stop bit, this used as a data integrity check. This is often referred to as a data frame [17].

As shown in Figure 2.7, five different parity bits can be used, the mark parity bit is always set at a logical 1, the space parity bit is always set at a logical 0, the even parity bit is set to logical 1 by counting the number of bits in the word and determining if the result is even, in the odd parity bit, the parity bit is set to logical 1 if the result odd.

The later two methods offer a means of detecting bit level transmission errors. Note that you don't have to use parity bits, thus eliminating 1 bit in each frame, this is often referred to as non parity bit frame.

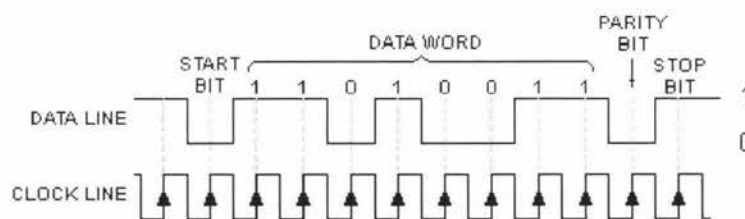


Figure 2.7 Asynchronous Serial Data Frame (8E1)

2.5.6. Bit Rates

Another important part of every asynchronous serial signal is the bit rate at which the data is transmitted. The rates at which the data is sent is based on the minimum speed of 300 bps (bits per second), the user may find some slower speeds of 50, 100 and 150 bps, but these are not used in today's technologies.

Faster speeds are all based on the 300 bps rate, you merely double the preceding rate, so the rates are as follows, 600, 1200, 2400, 4800, 9600, 19200 and 38400 which is the fastest speed supported by today's BIOS's. Note that a few years ago the fastest speed was of 19200 bps, because of all the strain exercised on the CPU because of the software control used to control the serial port. Today with the new Micro Channel, EISA, VL Bus and PCI motherboards, the new systems take advantage of bus mastering DMA control which has pushed rates up to 38400 by eliminating microprocessor overhead.

2.5.7. UARTS

Universal Asynchronous Receiver Transmitters (UARTs) are serial chips on PC motherboard (or on an internal modem card) [17]. The UART function may also be done on a chip that does other things as well. On older computers like many 486's, the chips were on the disk IO controller card. Still older computer have dedicated serial boards.

The UART's purpose is to convert bytes from the PC's parallel bus to a serial bit-stream. The cable going out of the serial port is serial and has only one wire for each direction of flow. The serial port sends out a stream of bits, one bit at a time. Conversely, the bit stream that enters the serial port via the external cable is converted to parallel bytes that the computer can understand. UARTs deal with data in byte sized pieces, which is conveniently also the size of ASCII characters [13].

For example when the user have a terminal hooked up to a PC. When the user types a character, the terminal gives that character to its transmitter (also a UART). The transmitter sends that byte out onto the serial line, one bit at a time, at a specific rate. On the PC end, the receiving UART takes all the bits and rebuilds the (parallel) byte and puts it in a buffer.

Along with converting between serial and parallel, the UART does some other things as a by-product (side effect) of its primary task. The voltage used to represent bits is also converted (changed). Extra bits (called start and stop bits) are added to each byte before it is transmitted. Also, while the flow rate (in bytes/sec) on the parallel bus inside the computer is very high, the flow rate out the UART on the serial port side of it is much lower. The UART has a fixed set of rates (speeds), which it can use at its serial port interface.

2.5.8. Interrupts

When the serial port receives a number of bytes (may be set to 1, 4, 8, or 14) into its FIFO buffer, it signals the CPU to fetch them by sending an electrical signal known as an interrupt on a certain wire normally used only by that port. Thus the FIFO waits for a number of bytes and then issues an interrupt [17].

However, this interrupt will also be sent if there is an unexpected delay while waiting for the next byte to arrive (known as a timeout). Thus if the bytes are being received slowly (such as someone typing on a terminal keyboard) there may be an interrupt issued for every byte received. For some UART chips the rule is like this: If 4 bytes in a row could have been received, but none of these 4 shows up, then the port gives up waiting for more bytes and issues an interrupt to fetch the bytes currently in the FIFO. Of course, if the FIFO is empty, no interrupt will be issued.

Each interrupt conductor (inside the computer) has a number (IRQ) and the serial port must know which conductor to use to signal on. For example, ttyS0 normally uses IRQ number 4 known as IRQ4 (or IRQ 4). A list of them and more will be found in "man set serial" (search for "Configuring Serial Ports"). Interrupts are issued whenever the serial port needs to get the CPU's attention. It's important to do this in a timely manner since the buffer inside the serial port can hold only 16 (1 in old serial port) incoming bytes. If the CPU fails to remove such received bytes promptly, then there will not be any space left for any more incoming bytes and the small buffer may overflow (overrun) resulting in a loss of data bytes. There is no Flow Control to prevent this.

Interrupts are also issued when the serial port has just sent out all 16 of its bytes from its small transmit buffer out the external cable. It then has space for 16 more outgoing bytes. The interrupt is to notify the CPU of that fact so that it may put more bytes in the small transmit buffer to be transmitted. Also, when a modem control line changes state an interrupt is issued.

Interrupts convey a lot of information but only indirectly. The interrupt itself just tells a chip called the interrupt controller that a certain serial port needs attention. The interrupt controller then signals the CPU. The CPU then runs a special program to service the serial port. That program is called an interrupt service routine (part of the serial driver software). It tries to find out what has happened at the serial port and then deals with the problem such as transferring bytes from (or to) the serial port's hardware buffer.

This program can easily find out what has happened since the serial port has registers at IO addresses known to the serial driver software. These registers contain status information about the serial port. The software reads these registers and by inspecting the contents, finds out what has happened and takes appropriate action.

2.5.9. IO Address & IRQ

Since the computer needs to communicate with each serial port, the operating system must know that each serial port exists and where it is (its I/O address). It also needs to know which wire (IRQ number) the serial port must use to request service from the computer's CPU. It requests service by sending an interrupt on this wire. Thus every serial port device must store in its non-volatile memory both its I/O address. Interrupt ReQuest number: IRQ. For the PCI bus it doesn't work exactly this way since the PCI bus has its own system of interrupts. But since the PCI-aware BIOS sets up chips to map these PCI interrupts to IRQs, it seemingly behaves just as described above except that sharing of interrupts is allowed (2 or more devices may use the same IRQ number).

I/O addresses are not the same as memory addresses. When an I/O address is put onto the computer's address bus, another wire is energized. This both tells main memory to ignore the address and tells all devices which have I/O addresses (such as the serial port) to listen to the address to see if it matches the devices. If the address matches, then the I/O device reads the data on the data bus.

2.6. Software Development Process

2.6.1. Introduction

As indicated in the previous section, CIM is one of the most discussed topics in manufacturing. CIM as the name suggests, in an effort to integrate activities in manufacturing through the medium of computers [12]. It is now widely accepted that CIM leads to widespread improvement in productivity. The CIM strategy is based on the successful integration of both hardware and software. Therefore, this section discusses some of the major issues related to software aspect of CIM and how Object Oriented (OO) concepts may be used to offer generalized solutions.

2.6.2. Objects & Object-Oriented Software

Object-oriented (OO) computing is a style of computing in which data and associated procedure are encapsulated to form an 'object'. Thus, an object is computational entity that exists at a higher level of abstraction than data structures or procedures. The term encapsulation implies that the data can be accessed only through pre-defined procedure. The other main principle in this paradigm is that objects communicate with each other can only by exchanging messages.

Software systems that implement OO techniques are popularly called Object-oriented programming systems (OOPS) or Object-oriented software (OOS). The OO paradigm is an outcome of the evolution of modularity concepts developed to improve various aspects of the software development life cycle. Most of these developments retained the traditional fundamental notion that

$$(Computer) Programs = Data + Algorithm (Procedures)$$

The basic problem with this approach is that most real world entities are encapsulations of data and procedures that characterize their behaviour. Therefore, a programmer or analyst must go through a transformation and often a restructuring process to construct programs to model the entities.

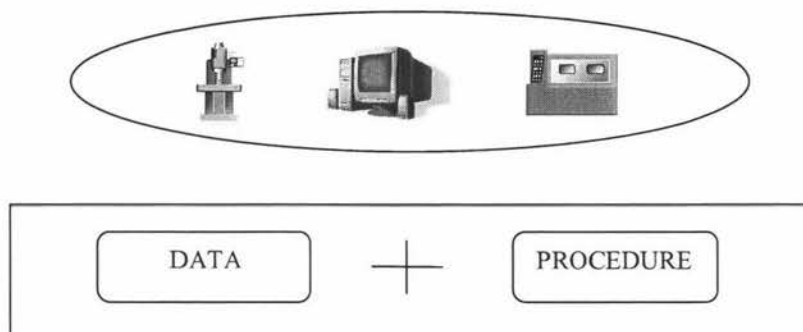


Figure 2.8: Traditional modularity concepts [12]

The OO paradigm represents a different way of looking at the program modules. It defines program modules as package of data and procedures named an 'object': i.e. an abstraction of private data and operations that are naturally associated together. Because of this abstraction facility, it is possible to represent real life factory objects such as N machines or a part quite close to reality (Figure 2.9).

Data is stored in locations, i.e. instance variables, which cannot be directly accessed by other objects. Procedures are commonly known as 'methods'. Each procedure (or method) defines the behaviour expected of the object. Such behaviour is to change the data stored in its instance variable. Objects interact by sending one another messages. That is, an object sends a message to another when it wants certain services from the object receiving the message.

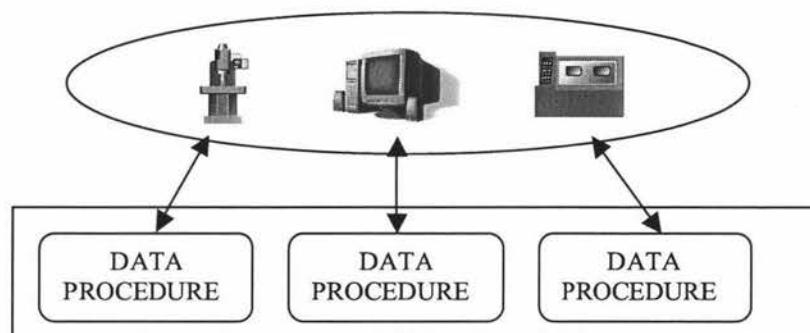


Figure 2.9: OO Approach [12]

There is a number of high level programming languages such as Borland's Delphi, Visual Basic, Visual C++, and Borland's C++. Because each programming language has its own programming features and characteristics, it is necessary to evaluate all the possible options and relevant requirements before choosing a suitable programming tool that will facilitate the development of the project.

The following are the requirements defined in order to select the most appropriate programming language [12]:

1. The language must be able to build a scalable, maintainable and reusable within the Rapid Application Development (RAD) environment.
2. The language must communicate with other Window based applications.

3. The application built with the language must be an open system application that fully utilise Window resources and keep in consistency with the graphic format from the Microsoft Window's Operating systems.
4. The application should be capable of creating a client server application.

2.6.3. Borland's Delphi 4.0

Based on all research conducted. Borland's Delphi is easier to learn for beginners, it also full meets the requirements listed above. As described by Reisdorph [10], the first version of Delphi was developed by Borland in 1994, and it is also Borland's best-selling rapid application development product for writing Windows applications. With Delphi, it is possible to write Windows programs such as Win 32 console applications or Win32 graphical user interface (GUI) programs more quickly. When creating Win32 GUI applications with Delphi, the user has the programming power of a true compiled programming language (Object Oriented Pascal) within a RAD environment. Hence, the user can create a GUI to a program using drag-and drop techniques for true rapid application development [10].

One of the most important concepts within Delphi is the Integrated Development Environment (IDE). As illustrated in Figure 2.10, Delphi IDE is divided into three parts. The top window can be considered the main window. It contains the toolbars and the component palette. The Delphi toolbar allows the usual one-click access to functions such as opening, saving, and compiling projects. The Component palette contains a wide array of components that the user can drop onto the main form. These components include text labels, edit controls, list boxes, buttons, and ActiveX controls.

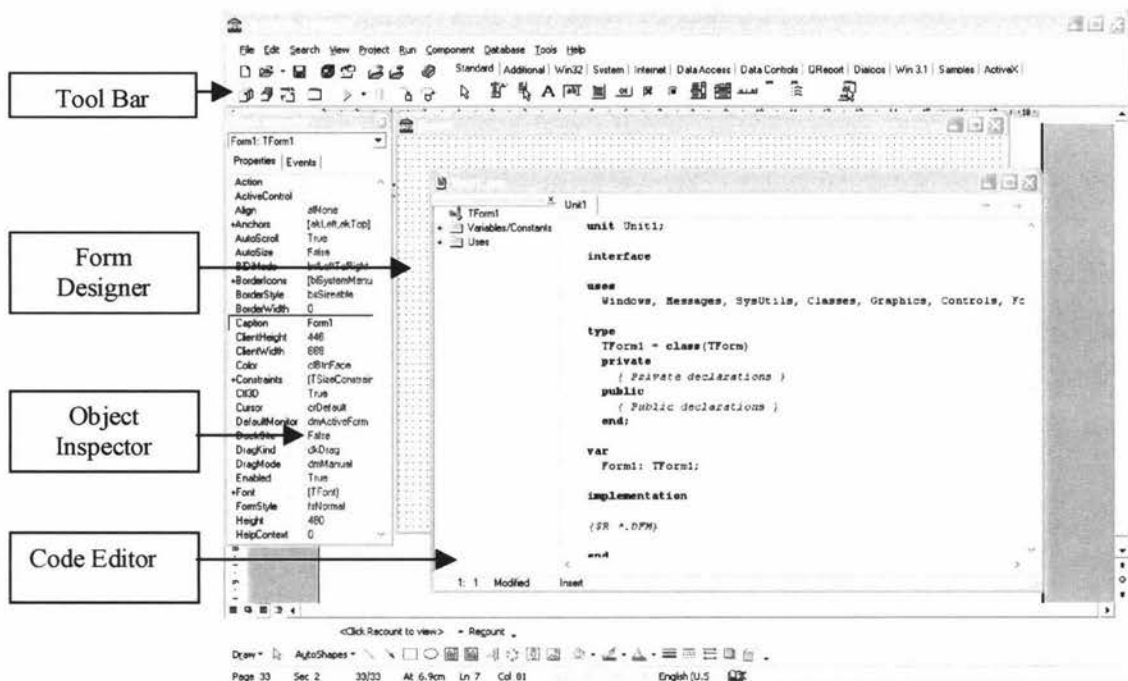


Figure 2.10: Delphi Interface

Object Inspector

Below the main window and on the left side of the screen is the object inspector. It is through the object inspector that a user can modify a component's properties and events. As illustrated in Figure 2.10; the object inspector has two tables: the properties tab and the events tab. A component's properties control how the component operates. For example, changing the Colour property of a component changes the background colour of that component. The list of properties available varies from component to component, although components usually have several common elements such as Width and Height properties.

The event tab contains a list of events for a component. Events occur as the user interacts with a component. For example, when a component is clicked, an event is generated that informs the user that the component was clicked. The user then could write code that responds to these events, performing specific actions when an event occurs. As with properties, the event could also be made responsive to variation from component to component.

The Delphi Workspace

The main part of the Delphi IDE is the workspace. The workspace initially displays the form designer. It allows the user to create forms, such as the program's main window, a dialog box, or any other type of window. The form designer will enable the user to place, move, and size components as part of the form creation process.

Hiding behind the form designer is the code editor. The code editor is where the user could type code when writing the programs. The object inspector, form designer, code editor and component palette work interactively as the user builds the applications [10].

CHAPTER 3

3. Multiport Card Installation

3.1. Introduction

As part of DNC system developed within this research project. It involves the study of a multiport board that is most suitable for the small & medium- sized manufacturers in New Zealand.

MS-DOS and the original PC BIOS were originally developed to support just two or perhaps four RS-232 ports. Users immediately saw the need for applications using 4, 8 or even 16 ports. Third-party manufacturers quickly responded to the demand and began producing nonintelligent multiport boards.

3.2. Background

These nonintelligent multiport boards typically contain an array of 8250 UARTS that can be configured to reside in various slots on the I/O memory bus. Each of the UARTs appears to the CPU to be more or less identical to a standard RS-232 port, with one important exception rather than have each UART directly control an interrupt line, all of the UARTS on the board share a single interrupt line. The board itself supplies hardware logic that manages and performs arbitration between all of these UARTs competing for use of a single interrupt line.

Figure 3.1 shows the Smartio C104 board selected for this research project. The board can be configured with either four to eight 16550 class UARTS. In this particular board, the C1104 is configured with four UARTS. Moxa has a standard extender cable that connects to the card edge via a large D Connector, and has four RS-232 cables extending from it. Each of the cables supports the standard signals used on PC RS-232 ports.



Figure 3.1: Moxa multiport card

3.3. Hardware Installation

The installation of the Smartio C104 series consists of hardware and software installation. The hardware installation is detailed in this section of the chapter, while the next section deals with the procedure involved in installing the software.

3.3.1. Quick Hardware Installation

The Smartio card features flexible hardware configuration that allows a quick and easy method installation for users. The procedure firstly requires the user to short the jumper JP1 on the upper left corner of the board. As illustrated in Figure 3.2 the card is inserted into the desired PCI slot. The user could now proceed to the software installation phase. Once the software installation is completed, the user then need to cold start the system to complete the quick installation.

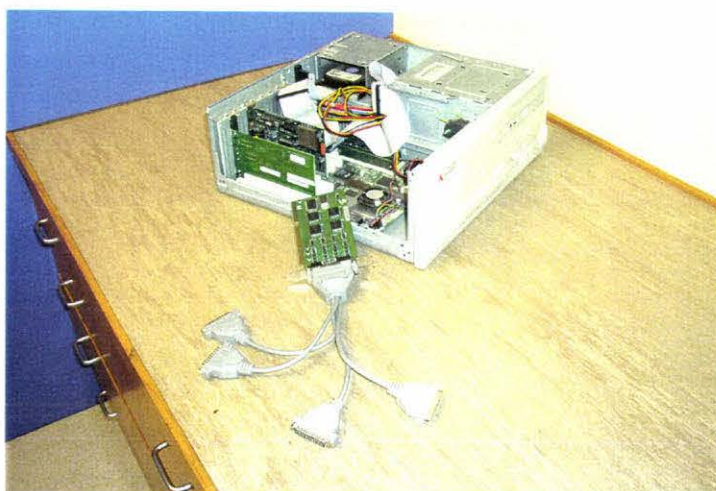


Figure 3.2: Inserting the Multiport Card into the PCI Slot

3.3.2. Hardware Installation with IO-IRQ Utility

Although the quick installation seems very simple and efficient. This quick installation process did not activate the C104 card during the course of this research project. Therefore, the IO-IRQ Utility program was used instead.

The application IO-IRQ utility shown in Figure 3.3 comes after the C104 is plugged into the PCI slot in the PC. The user must run the IO-IRQ.exe utility in the driver diskette under DOS system to change the hardware configuration.

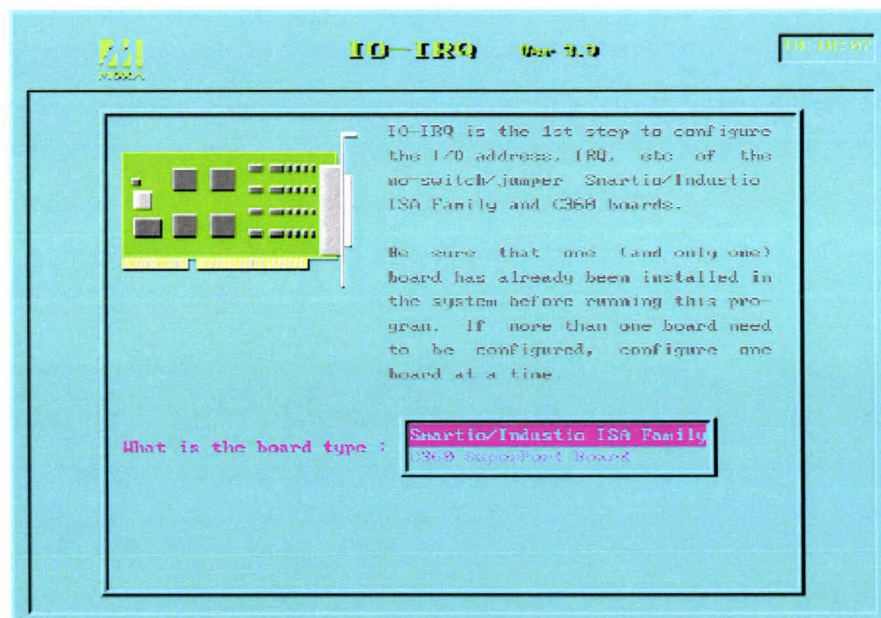


Figure 3.3: IO-IRQ utility

Once the user selects the “Smartio/Industio ISA family on the interface, it was then possible to re configure the CAP address of the C104. The values for the default setting is listed below

<p>I/O Address: 0x180 (Port1), 0x188 (Port 2), 0x190 (Port 3), 0x198 (Port 4)</p> <p>IRQ: 10</p> <p>INT Vector: 0x1C0</p> <p>Configuration Access Port (CAP): 0x180</p>

3.4. Software Installation

This section is a follow on from the previous section on hardware installation. It illustrates areas such as software driver installation; configuration and driver update procedure described for the selected Windows NT operating system. Windows NT is still by far the most used operating system within the manufacturing sector. Characteristics such as stability, network integration, and mobile computing make this OS a common application within the manufacturing shop floor. Windows NT supports up to 256 serial ports from COM1 to COM 256. To fully integrate the advanced features of Windows NT, multi-process and multi-thread, pure 32-bit Windows NT device driver are developed for the C104 multiport boards.

3.4.1. Installing Driver

In order to install the driver, the user must first log in as the Administrator. Once logged in, the user must access the control panel and select the network icon to activate the adaptor tab. Figure 3.4 shows the driver can be located in the driver diskette, where a file by the name of WINDOWS.NT will activate the installation/configuration process.

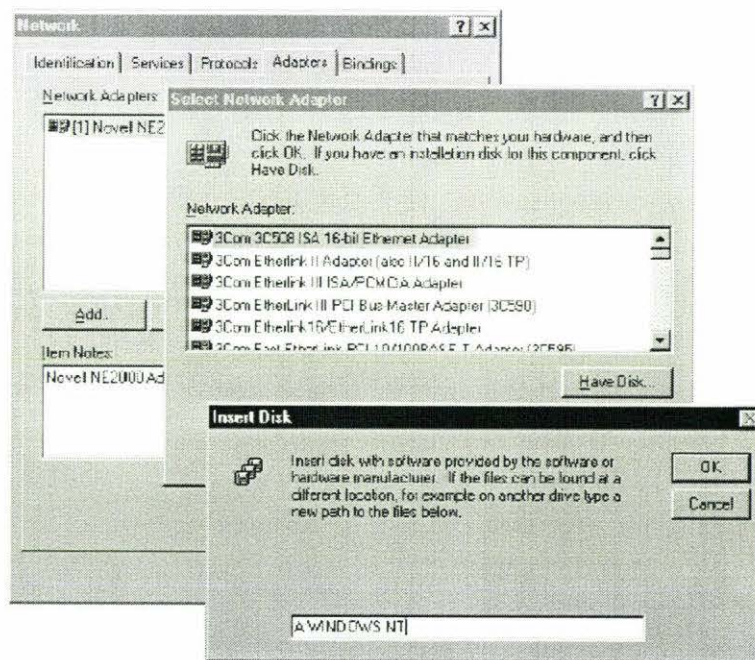


Figure 3.4: Driver installation

The installation process then presents the user with the Moxa Smartio/Industio Configuration panel dialog box, where the user could configure the board by configuring the properties. By clicking on the add button shown on Figure 3.5, the user can enter the property dialog shown on Figure 3.6, where the user needs to select the "C104 Series" in the "Board Type" field. If necessary, the user also needs to type the desired interrupt vector address, in the "INT Vector" field. By selecting the desired interrupt number in the "Interrupt No" field and typing the desired base I/O address, in the "Base I/O Port Address" field. All the settings should then match settings that are physically set on the board and conflict with no other devices.

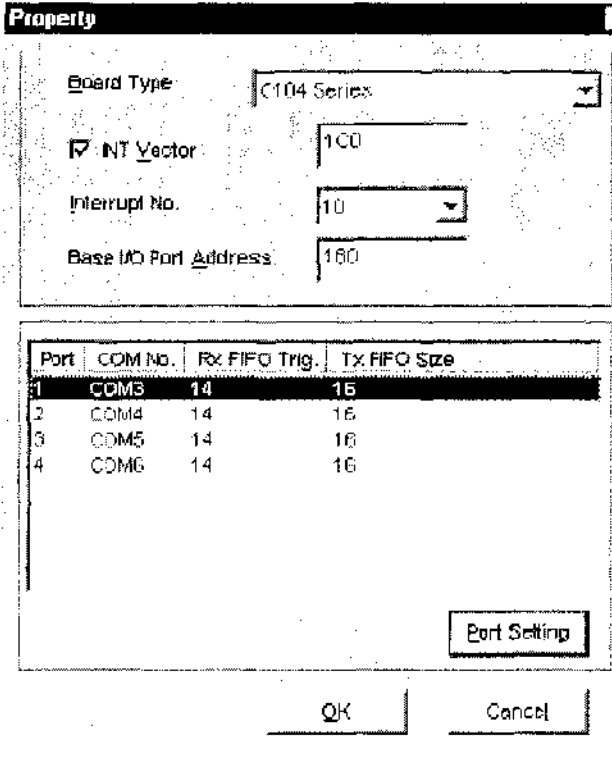


Figure 3.6: Property Dialog

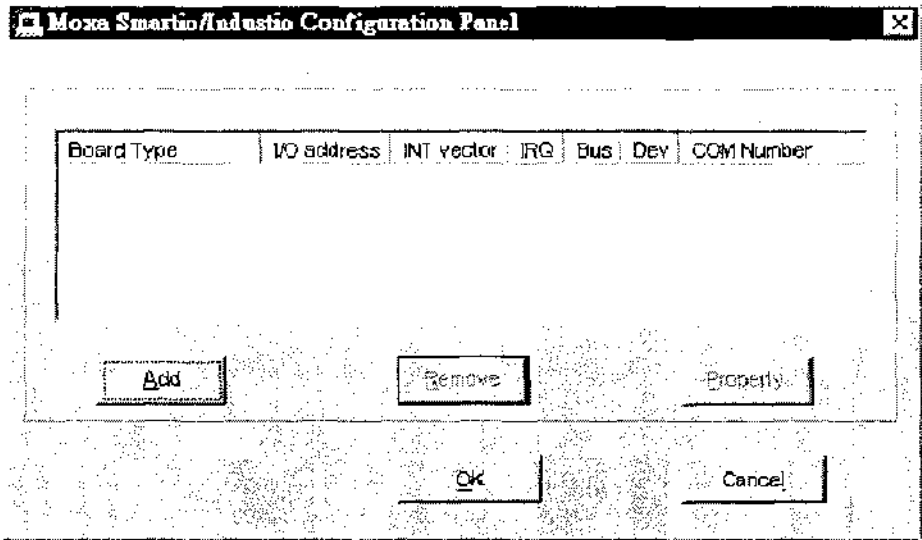


Figure 3.5: Configuration Panel

3.5. Serial Programming Tools

As part of the technical support offered from Moxa, PComm, a professional serial communication tool for PC is included with the Smartio C104 multiport card. It consists of powerful serial communication library for easy programming in most popular languages, and useful utilities such as diagnostic, and terminal emulator.

The serial communication library is commonly used during the development phase of data communication, remote access, data acquisition or industrial control in the Windows operating systems.

3.5.1. Diagnostic Tool

As one of the supporting utility provided by Moxa. A convenient diagnostic program such as "PComm Diagnostic" shown in Figure 3.7, provides internal and external testing of the multiport. This program offers testing for of IRQ, TxD/RxD, UART, CTS/RTS, DTR/DSR for the C104 board and ports to verify correct operation of both the software and hardware.

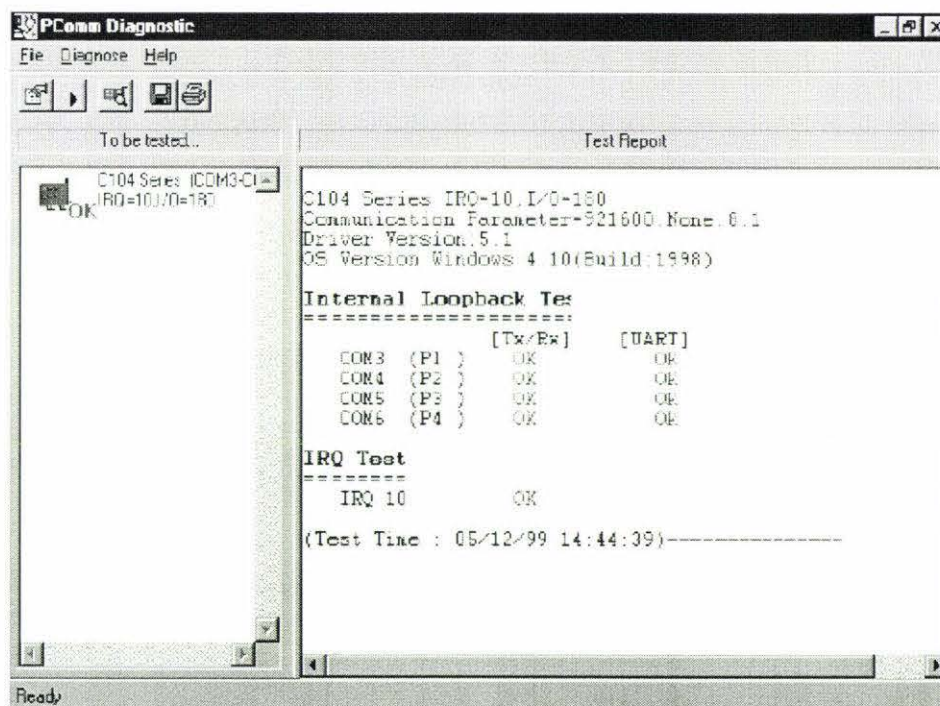


Figure 3.7: Diagnostic software

This utility was used immediately after the C014 board was installed. This utility was proven to be effective when the first quick installation failed on the workstation used. Thus enabling quick debugging of the installation problem, and resolving the problem by implementing the IO-IRQ installation method.

3.5.2. Terminal Emulator

Once the system have been checked by the diagnostic program. It was then possible to make use of the terminal emulator program that is included in the PComm serial programming tool. The terminal emulator shown in Figure 3.8 was used as a benchmark during the development of our own data communication software. Terminal emulator features multi-windows and supports terminal type VT100, ANSI and Dumb. It is possible for the user to transfer data interactively, send pattern periodically and transfer files using Zmodem, Ymodem, Xmodem, Kermit or ASCII protocols.

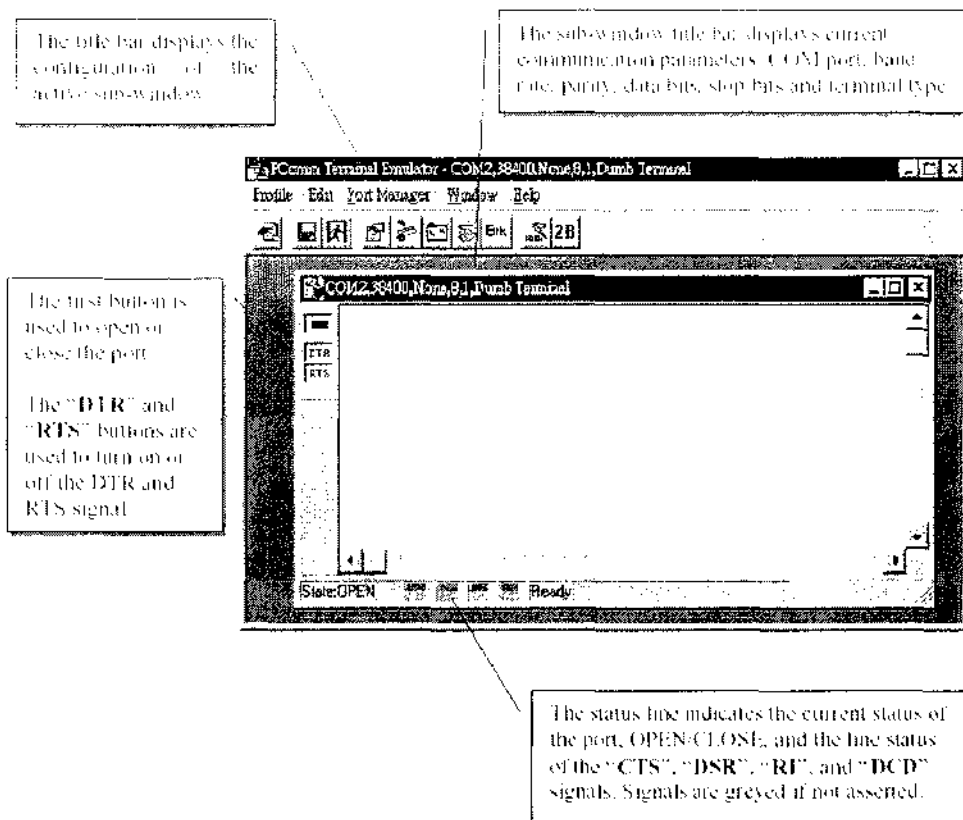


Figure 3.8: Terminal Emulator

To start a terminal emulation session on a COM port, the user must first ensure the RS-232 cables are connected correctly (see next section on cabling) and the setting in both of the terminal are adjusted to the correct setting using the port setting dialog shown in Figure 3.9.

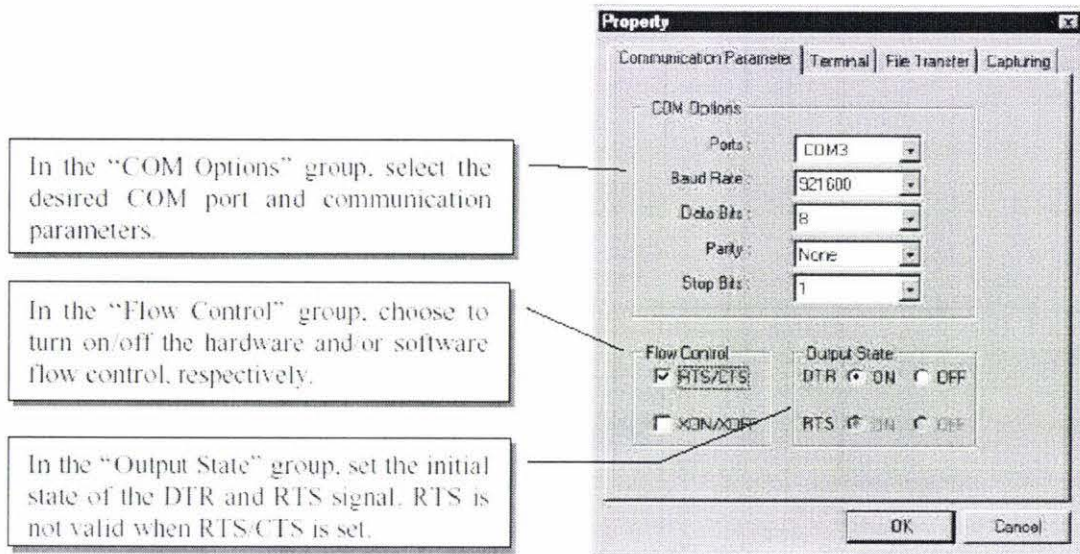


Figure 3.9: Setting dialog

With the correct setting, the terminal emulator allows the user to carry out versatile operations such as file transfer, or sending pattern between terminals.

3.5.3. Data Scope

After the system was set up in the PC. An immediate test was conducted out to examine whether the terminal emulation program was fully functional. During the testing stages, minor problems were encountered the configuration of the cables. As part of the approach taken in pin pointing the source of the problem was through the application of the Data scope utility within the serial communication tools. The Data Scope is also a utility program that assists the user with serial communication trouble shooting and debugging for any Win32 compliant COM port. It offers transparent monitoring capability of serial communication lines and allows data to be streamed to disk storage for later analysis.

This utility has the advantage of turning a simple PC or notebook into an economical but powerful data analyser for serial communications. By using two COM ports, it is possible to easily tap into two serial devices at the RS232 interface and watch all the data and communication status between the two devices.

To in order to activate the data scope, the user must first configure the port to the desired connection mode. The user must decide whether the scope will use Transparent mode (Figure 3.10) or Bridge mode (Figure 3.11).

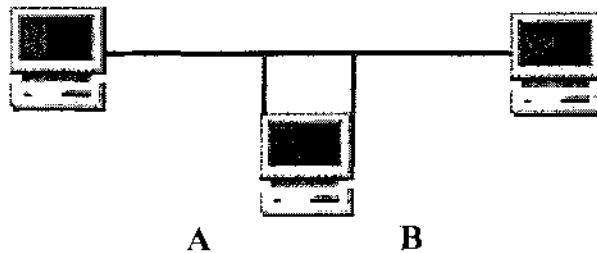


Figure 3.10: Transparent Mode

Under transparent mode, data scope will transparently monitor the data and line status across the communication line. Port A will listens to the data sent by Device A and Port B will listen to the data sent by device B.



Figure 3.11 Bridge Mode

Under the bridge mode, data scope will transmit the data received from Port A to Port B, and vice versa, but not including the line status signals. The Device A and Device B are physically connected to the PC running data scope utility with two RS-232 null modem cables

During the testing session, the bridge mode was used, and the result from the test was ascertained and illustrated in the following Figure 3.12.

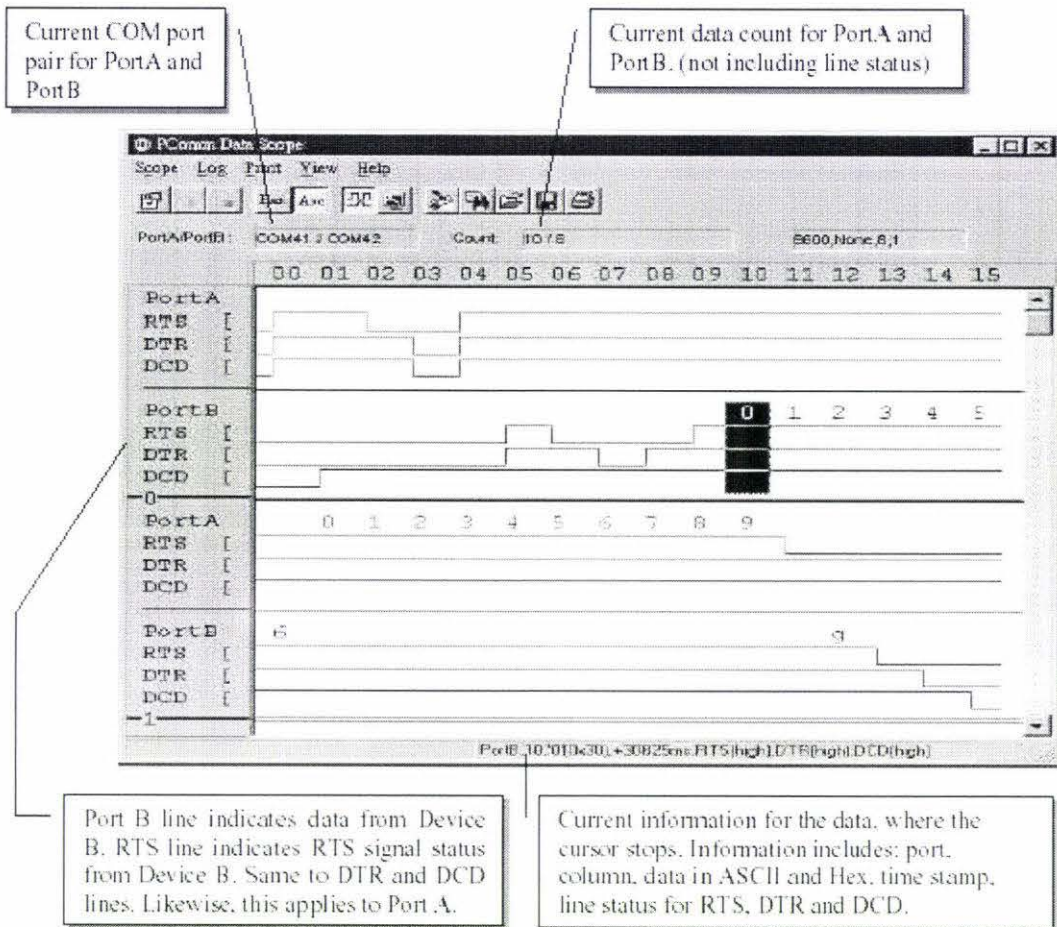


Figure 3.12: Data Scope

3.6. Cabling

After the C104 card was installed in the PC. The next task was to gather the various cables required for the RS-232 connections. The connection for the C104 is a 25 pin male connector. However, most serial ports are 9 pin female connectors, and during the initial stages of the research project, the following cables and adaptor were used for the serial communication to take place.

3.6.1. 9 Pin Female to Female Gender Changer

Figure 3.13 shows the standard “Female Gender Changer” cable. This cable has a female D-subminiature 9-pin connector on one end and its female equivalent on the other. All the signals are routed straight through, terminating on the same numbered pin where it began.

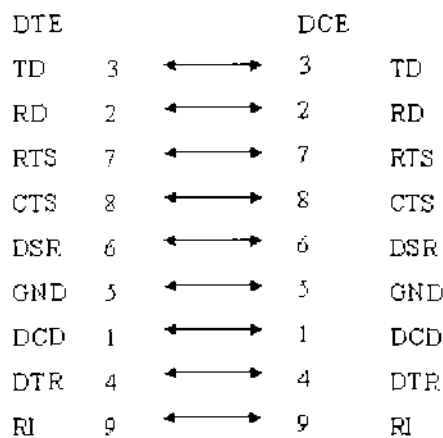


Figure 3.13: 9 Pin gender changer

3.6.2. The Null Modem Cable

This cable is required when test are carried out between PCs. The standard PC-to-modem cable does not work because both PC are DTE devices, both will transmit on Pin 2 and receive on Pin 3 of their 25-pin connectors. Thus, a cable that routes all signal straight through will be connecting the output from one PC directly to the output to the other PC, and the RD line from PC will be connected to the RD line of the other.

A Null modem cable that crosses over the pins, with exception to the ground pin was used. Figure 3.14 shows the wiring diagram used for this operation.

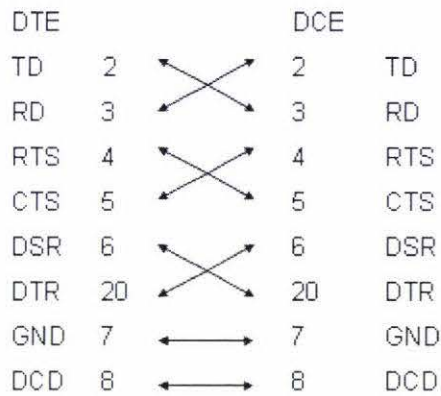


Figure 3.14: Null modem cable

3.6.3. Installing the Cable

Once the necessary cables and adaptors are gathered, it was then possible to link to the testing jig PC to the C104 board. The Figure 3.15 below illustrates all the cables and adaptors used in making this connection.

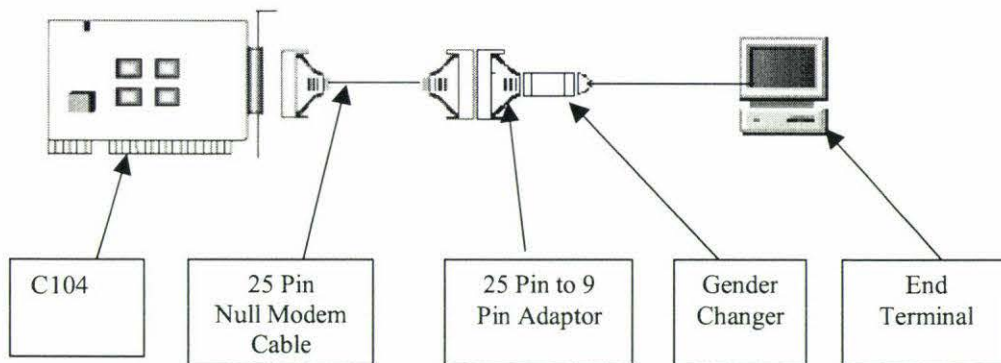


Figure 3.15: The entire cable and adaptors used

3.7. PComm Application Programming Interface (API)

To understand how to communicate and transfer data between the terminals. It is important to understand some of the basic fundamentals of API.

The acronym API stands for Application Programming Interface. It serves as a software interface to be used by other programs. Just as a number keypad is the interface for a calculator. API is the set of classes, functions, and methods of a particular programming language. Developers use the API to code the software, and it has the ability to simplify commands sent to the operating system or computer hardware.

PComm provides API libraries for establishing communication with the multiport and end terminal. Its purpose is to assist users to develop programs for serial communications for any COM port complying with Microsoft. It can ease the implementation process of Multi-process and multi-thread serial communication program and hence greatly reduce the developing time. It is suitable for all Win 32 compatible COM ports. The hierarchical diagram in Figure 3.16 shows the PComm library.

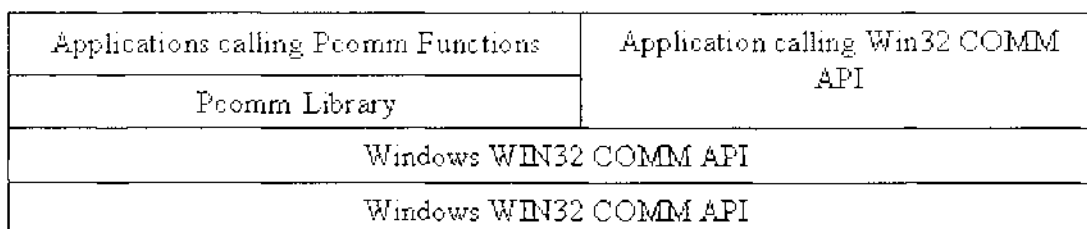


Figure 3.16: Hierarchical PComm library

The PComm library functions can be divided into categories such as Port control, data input/output, and file transfer.

CHAPTER 4

4. DNC Software Development Process

4.1. Introduction

This chapter describes the software development process for the system. The software is developed through the Waterfall model illustrated in Figure 4.1, and the 4 stages of progression are as follows:

Requirement Analysis & Definition

This stage investigates the system's services, constraints and goals. The final system requirements are established by consulting with system users.

Interface Design

The interface design process partitions the requirement of the software system. It establishes overall system architecture.

Implementation

This involves the actual coding of software sub-systems that have been analysed and designed, and the integration of these sub-systems into a complete functional system. Due to the structure of the chapters, the detailed description of the implementation process is described from chapter 5 to chapter 10.

Testing

Testing stage will be carried out when a fully designed and implemented module is completed. Sets of exercises will be designed to test the performance of the application. Through these testing, software functions will be accurately verified and conformed to the specification.

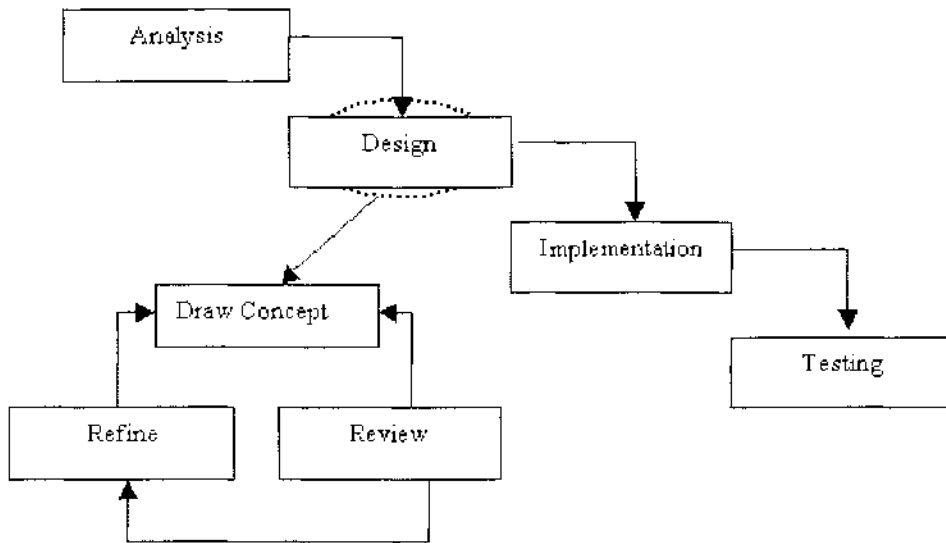


Figure 4.1: Waterfall model

4.2. Requirement Analysis & Definition

4.2.1. End- User/Operating Environment Analysis

The primary purpose of structured analysis is to model the system's services, constraints and goals by consulting with target users. When developing a GUI application, the analysis activity focuses on the user and the user's tasks. To develop user profiles, the analysis looks at the user's frequency of use and the user's tasks.

People who come in contact with the use of this system are usually operators within an industrial environment that need to transfer design files or NC programs through the RS232 network on a day to day basis. It can be assumed that the users of this system are technically qualified NC machine operators that are experience with the use of computers. The amount of knowledge will vary from one to another, but it is assumed that most users have used computers in the Microsoft Windows environment.

Due to the fact that there are only a small amount of factories implementing advanced DNC system. The end users are not expected to have any knowledge of what a DNC system is or any prior experience in using similar packages.

A DNC structure built with a proprietary operating system will limit a manufacture in implementing CIM applications in the future. The strong domination of Microsoft Operating Systems (OS) over the past few years has made Windows a de facto standard within almost every sector of industry. Therefore software developed for this project runs under any operating system from Window 95 or above.

Environmental Attribute	Description
User type is novice to experienced	The minimum user knowledge is confidence with Microsoft Windows environment. However some users may have previous experience in NC programming.
DNC for commercial purposes	The software will be installed in small to medium manufacturers. The system will replace some of the manual program insertion and older generation systems.
The users technical CNC machine operators	The user of the DNC have been identified as operators in manufacturing environment
DNC is run under Microsoft Windows 9x/NT	The new system should not change the computer hardware requirements for the DNC

Table 4.1: Specifications

4.3.2. Design Concepts

From the specification above, the design process is design-by-refinement where a concept is drawn and then reviewed for suitability and adaptability.

Figure 4.2 shows storyboarding were used as the principle method of design lay out. First of all, the required task was documented in operations through OO program design approach. These operations were then expanded into a sequence of steps where each step was a designed element. To illustrate the idea, the following is an example of a very simple use of exchanging characters between the two terminals

1. Initiate the system.
2. Specify the terminals to initiate communication.
3. Activate the serial port by declaring the necessary conditions.
4. Start the communication with the desired terminal.

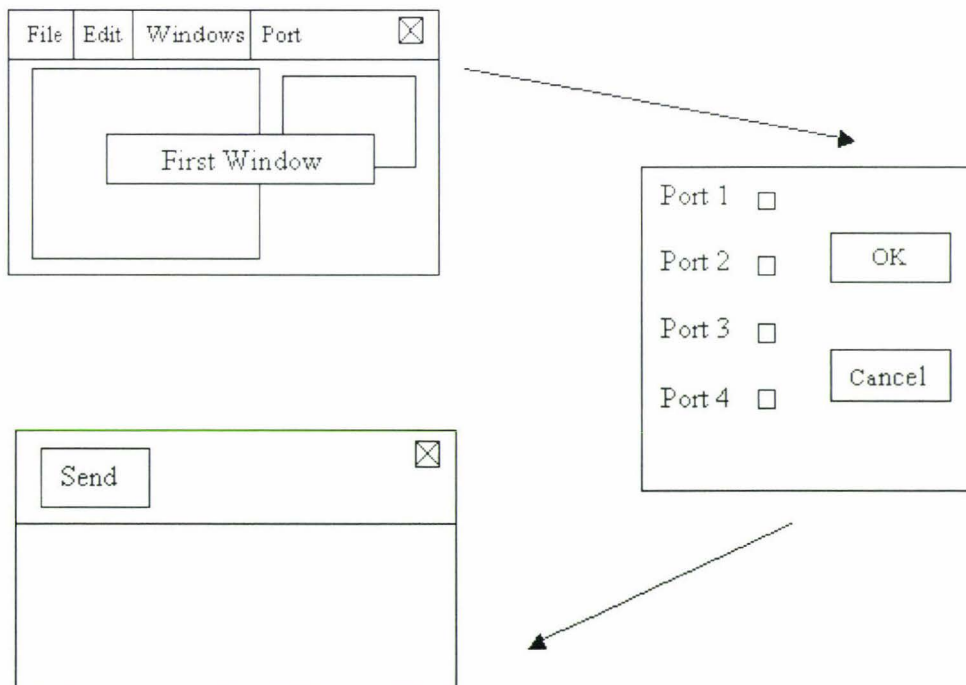


Figure 4.2: Storyboarding

4.3.3. Final Design

The combination of all those concepts and design iterations resulted in the final design that is illustrated in Figure 4.4. The final design was developed using Multiple Document Interface (MDI) technique. Many window-based applications such as Microsoft Excel, Microsoft Word, and CAD systems etc have similar multiple document interfaces. The interface consistency makes it easy to operate in the Windows environment. A MDI application allows users to display multiple documents at the same time, with each document displayed in its own window. Document windows are contained in a parent window, which provides a workspace for all the document windows in the application. As stated in the previous section, this system contains four independent modules. The communication model mainly takes care of data transmission and records NC programs downloaded and uploaded. Other relevant manufacturing information can also be viewed through this module. NC editor facilitates the users to edit or modify NC programs within the DNC system. The OLE MDI form will allow the users to link the system to other existing software on the desktop.

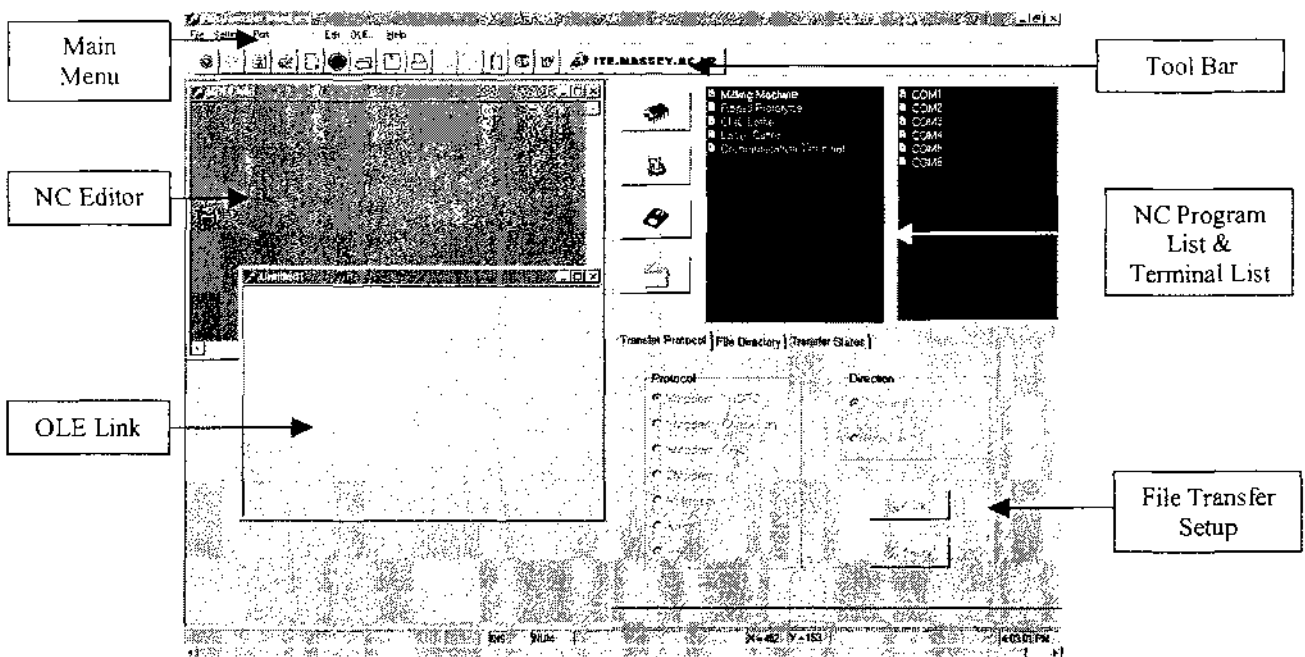


Figure 4.4: Final design of main interface

CHAPTER 5

5. Establishing Communication

5.1. Introduction

This section of the report explains the implementation phases of the program. Denoting areas such as the structure of the program and the actual development procedures of the system. It covers the programming techniques used, the application of Delphi programming functions and the step-by-step explanation on establishing communication through a multiport card.

5.2. Establishing Communication

The first step taken to establish a serial communication between the multiport card and a second terminal is to open a communication channel. This research begins by establishing a new project file called *FounTera.dpr* and a main form named *FtansM.pas*. In order to access the PComm Serial tool, two more files must be added to the project. Figure 5.1 illustrates how to add these files using the project manager in Delphi4.

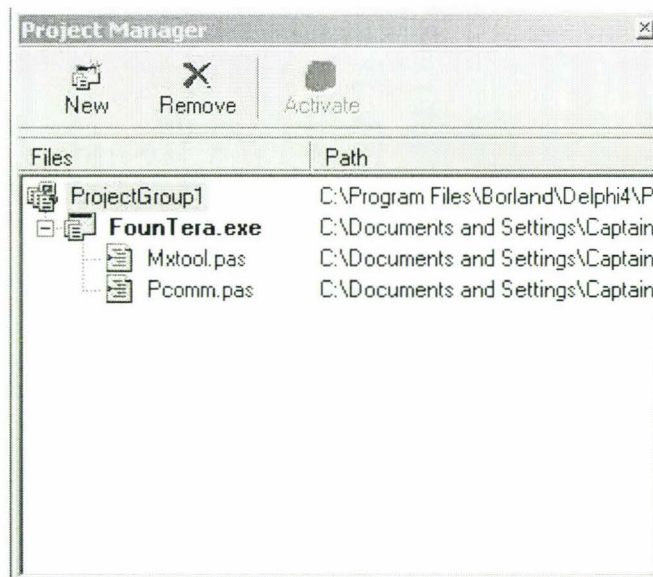


Figure 5.1: Delphi's project manager

5.2.1. PComm.pas

The full context of the PComm.pas can be found in section 10 of Appendix A. This file is the main file within the PComm serial programming API. The first part of this file has numerous declaration of constants for ports and file transfer settings. The rest of the PComm file imports routines from the PComm.dll and are all declared as functions within this file.

5.2.2. MxTool.pas

MxTool is a Pascal file that is within the PComm serial programming tool. This file is mainly responsible for the handling of error messages. The file has a specific procedure that lists out the main errors when they are encountered.

5.2.3. ExGlobal.pas

After those file for the PComm serial tool has been installed. The main interface *FtransForm* then needs a Pascal file which contains the Global variable & COM port record defined for the main interface. The *ExGlobal.pas* shown in the Appendix A starts by declaring:

```
type
  TExampleForm = class(TForm)
    Term: TMemo;
```

The line `TExampleForm = class(TForm)` describes a form created in the memory. The `TFtransFrom` inherits `TForm`'s members, a class that Delphi provides for creating forms. While `Term` is a Memo class that is also used later on during the program.

The next part of this program is a record named `TCOMMDATA`. A record is a Pascal syntax that allows a collection of related data to be rolled up into a single storage unit. In this program, `TCOMMDATA` is a single data variable that holds all fields needed in configuring a serial port.

For example,

```
TCOMMDATA = record
    Port : LongInt;
    BaudRate : Integer;
    Hw : boolean;
```

The next section of the *Exglobal.pas* declares the Global variables required in the *FtransForm*. As shown in Appendix A, the first variable is *GcommData*, it is declared so that instances can be created in *FtransM*. The second variable *GszAppName* is a string declaration that stores the name of the application, and the third variable declares the *GhForm* as a *TExampleForm*. The rest of the global variable declarations include numerous other array of variables required, for example, the following array declares all the 5 possible parity bits found in serial communication.

```
GstrParityTable :array[0..4] of string = (
    'None','Odd','Even','Mark','Space' );
```

The very last section of this file is the codes used for the procedure *ShowStatus*. Figure 5.2 illustrates the end effect of this *ShowStatus* function, this procedure writes the information on the COM port on top of the Window after a user chooses to open a serial port.

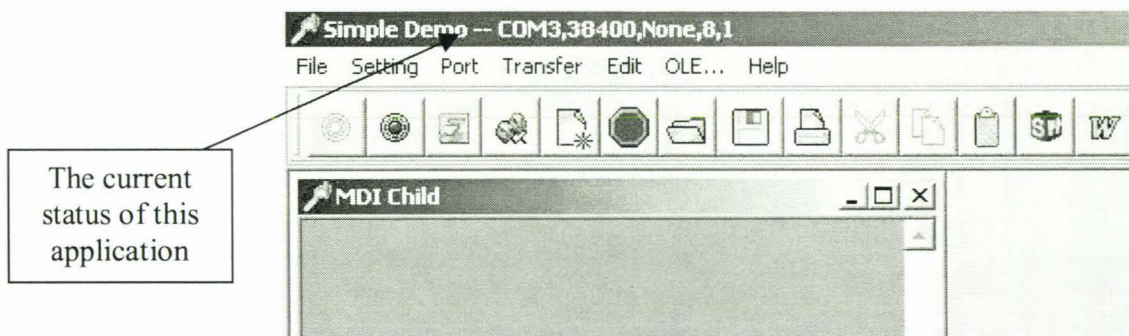


Figure 5.2: Showstatus

5.2.4. Config.pas

In order to adjust the setting of the serial port for establishing communication between terminals, the next step of the research project is to make a form that allows the user to adjust the setting of the serial port. Figure 5.3 illustrates what *config.pas* looks like

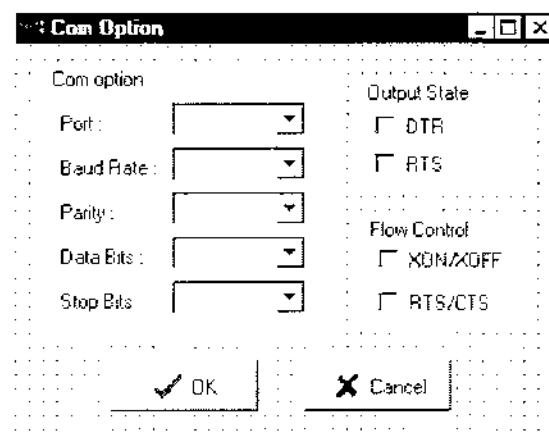


Figure 5.3: *Config.pas*

The name of this form is called a *CfgForm*, and contains 5 procedures that are used in manipulating the serial ports. The first procedure *FormCreate* is used specifically for configuring COM ports. As shown in section 5 in Appendix A, the *FormCreate* procedure uses a for loop function that adds the word COM to the start of every port.

The *FormActive* procedure allows all the selected setting from this interface to take effect when a port is opened. For example, the line `cbBaudRate.ItemIndex := ibaudrate` declares the baud rate to be the value selected from the Combo Box. The *chHwClick* procedure provides a simple technique that turns off the RTS output state when RTS/CTS is turned on (see Figure 5.4).



Figure 5.4: *Output state & flow control*

This is achieved by first declaring a Boolean variable called `Gfhw` in the start of the code, and if the RTS/CTS (`chHw`) is enabled, the RTS (`chRts`) will be automatically disabled.

The last two procedures, the `CfgCancelClick` and the `OKClick` are used for allowing the user to make the newly determined setting of the serial port to replace the default setting provided in the `FormCreate` procedure in `FtransM.pas`. If the `Ok` button is accidentally pushed. A pop up window will appear for the very last minute confirmation of the setting as shown in Figure 5.5.

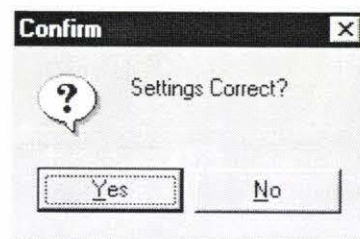


Figure 5.5: Confirmation Dialog

5.2.5. FormCreate Procedure

Having included the `PComm.pas`, `MxTool.pas`, `ExGlobal.pas`, and `Config.pas` it is now possible to construct the main interface. The first part of coding is the `FormCreate` procedure:

```
procedure TFTransForm.FormCreate(Sender: TObject);
```

As illustrated in section 1 of the Appendix A, the code of the `FormCreate` procedure includes many other lines of code that are used for other functions. However, this chapter is only interested in establishing communication between the multiport and the main PC terminal. Therefore, only functions, codes and programming techniques that are closely linked with the establishment of the terminal communication is discussed in this section

The first part of the FormCreate procedure uses a with statement to set up instances for the GcommData record. The use of With statement allows the declaration of the record without using record identifiers and dot operator. In this declaration, all the initial default parameter for the serial port is listed. As it is shown, if the user chooses to open up a serial port, the default opens with the following settings:

```

Port := 3;
ibaudrate := 14;
iparity := 0;
ibytesize := 3;
istopbits := 0;
BaudRate := B38400;
Parity := P_NONE;
ByteSize := BIT_8;
StopBits := STOP_1;
    
```

Figure 5.6 shows the default serial port declarations of the serial port after it is activated.

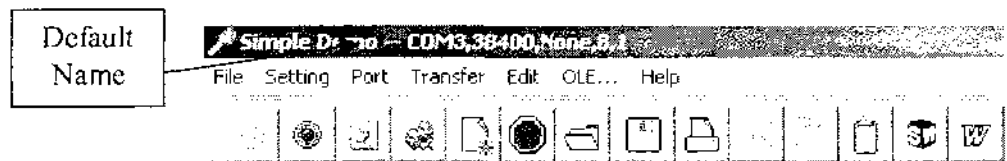


Figure 5.6: Default name

5.2.6. Menu

As shown in Figure 5.6, main menu is also a component that is being applied during the early phase of the programming. Every item in a menu is an object of the TmenuItem class; Delphi automatically creates these objects when the user designs menus with the Menu designer. As shown in figure 5.7, the menu designer can be sized in any way that the user desires, it also offers the ability to create short cut keys assignments for menu commands.

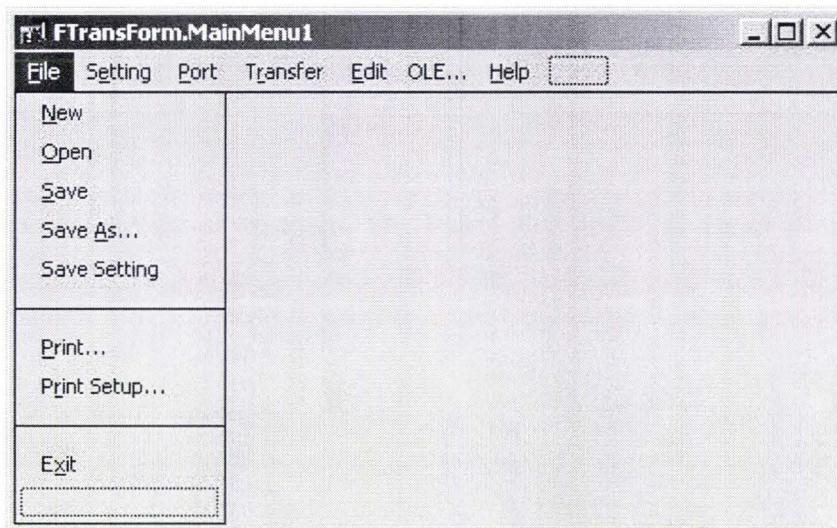


Figure 5.7: Menu Designer

The next relevant procedure is SwitchMenu procedure. This procedure inhibits the use of menu item when certain conditions are imposed, for example when a serial port is opened, the port open menu button will be automatically made inaccessible until the serial port is closed.

5.2.7. PortSet Function

The PortSet Function is a very important function used in setting up the serial port. Functions are very similar to procedures, both functions and procedures are executed when the interface needs to perform specific actions in a program. However, a function has to return a value, while procedure doesn't. As illustrated in Figure 5.8, an anatomy of a function can be dissected into following parts

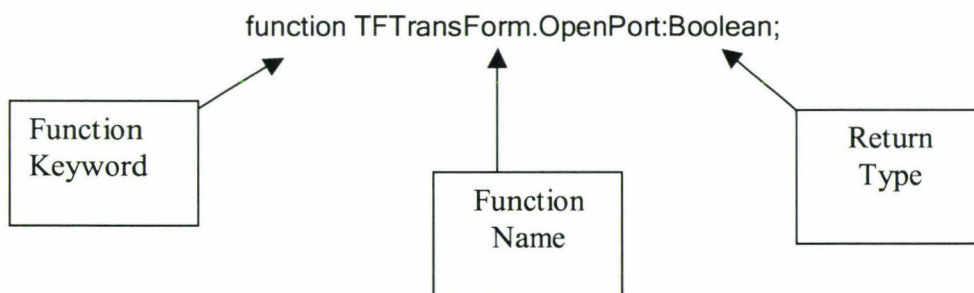


Figure 5.8: Anatomy of keyword Function

The `PortSet` function provides port control functions for all the serial ports. As shown in section 1 of Appendix A, `PortSet` function makes use of the APIs provided by `PComm` serial programming tool. The first API used here is `sio_ioctl`, this API configures communication parameter such as baud rate, parity, data bits and stop bits. This function requires argument such as `Port`, `baud` and `mode` as parameters, for example:

```
Port = COMPort number;
    { port := GCommData.Port;}
```

```
Baud=(bits/sec);
    { GCommData.BaudRate;}
```

```
Mode=bitcnt OR stop_bit OR parity
    {mode := GCommData.Parity or GCommData.ByteSize or GCommData.StopBits;}
```

`Sio_ioctl` returns `SIO_OK`, which is a long integer value represented by `ret`. If the output of `sio_ioctl` is not `SIO_OK`, the function will then activate the `Show Error` function provided by `MxTool.pas`. The rest of this function also applies similar coding algorithm, it checks setting for hardware and software flow control (`sio_flowctrl`), DTR state (`sio_DTR`) and RTS state (`sio_RTS`) for the serial port setting.

5.2.8. Open Port Function

The `OpenPort` function starts off by declaring `OpenPort` as `false`. It then uses a `sio_open` from `PComm.pas` API to allow a COM port to be opened for data transmitting and receiving. The `sio_open` function takes on the value for the COM port number, and returns `SIO_OK`, which is a long integer represented by `ret` in `FtransM.pas`. Another important aspect of the `OpenPort` function is that if the `PortSet` function is to be found `false` within the `OpenPort` function. The `OpenPort` function would then close the COM port and stop transmitting or receiving any data.

5.2.9. Setting1Click Procedure

The setting1click procedure is used to adjust the setting of the serial port, As shown in the Appendix A, when the setting is clicked using either the roll down menu or through the quick access button. The procedure will immediately access the CfgForm in *Config.pas*. As shown in this procedure, if the user chooses to cancel, the CfgForm disappears, and FtransForm takes the default setting declared early on in FormCreate procedure. If the user chooses the Ok button, the GcommData will then replace the default setting in FormCreate procedure.

5.2.10. PortOpenClick/ PortCloseClick Procedure

Having declared the OpenPort & ClosePort function previously, these functions can now be access through the roll down menu. The PortOpen procedure activates the OpenPort function, and will produce a “Beeping” sound. Both of these procedures will also show hints on the status bar when activated.

5.2.11. Testing for Signals

All the programming listed in this chapter is the fundamental code required to open and close port. To test whether these procedures have been written correctly, the user must use the data scope from the PComm serial programming tool to see if the serial port have been open or closed when the user pushes the menu buttons. In the research project, a device called a RS-232 tester was also used to provide an even quicker indication as to whether serial ports have been accessed successfully.

The following three settings were used to demonstrate the program works:

First Test:

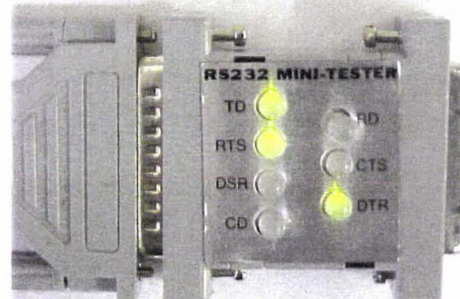
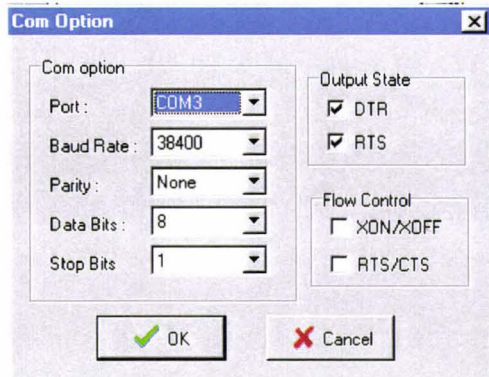


Figure 5.9: Test 1

Second Test:

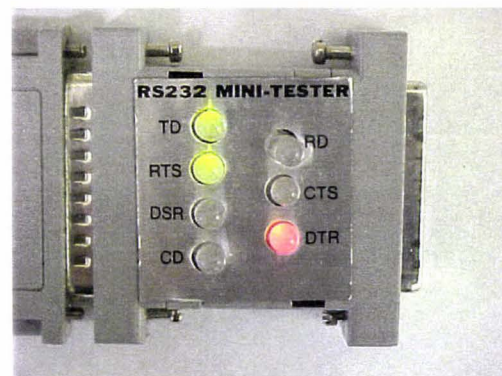
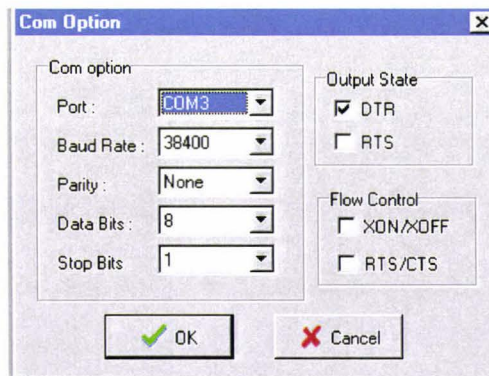


Figure 5.9: Test 2

Third Test

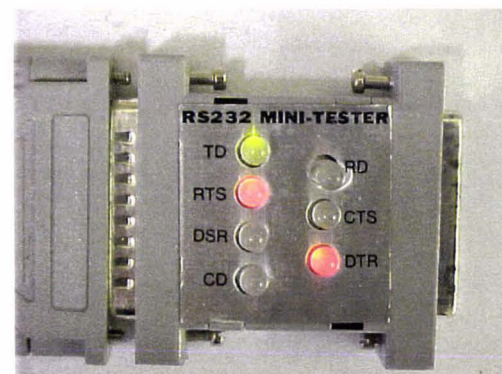
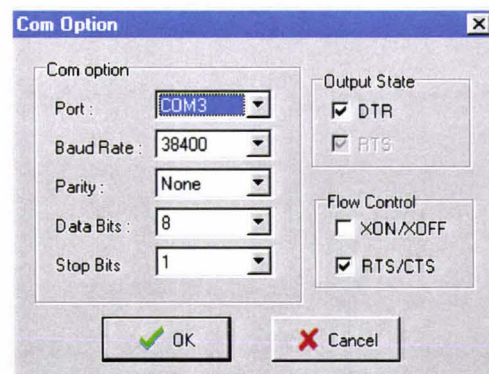


Figure 5.9: Test 1

CHAPTER 6

6. Character Exchange Interface

6.1. Introduction

This chapter describes how SimpleForm from *SimpleM.pas* transfer characters through the serial ports. Apart from the similar structure and programming skills used in *FtransM.pas*, threading technique was adopted to allow data to be displayed on the end terminal.

6.2. SimpleM.pas

As stated in the introduction, this file contains codes that are very similar to the procedures and functions found in the early sections of the *FtransM.pas*. The graphical interface is shown in Figure 6.1.

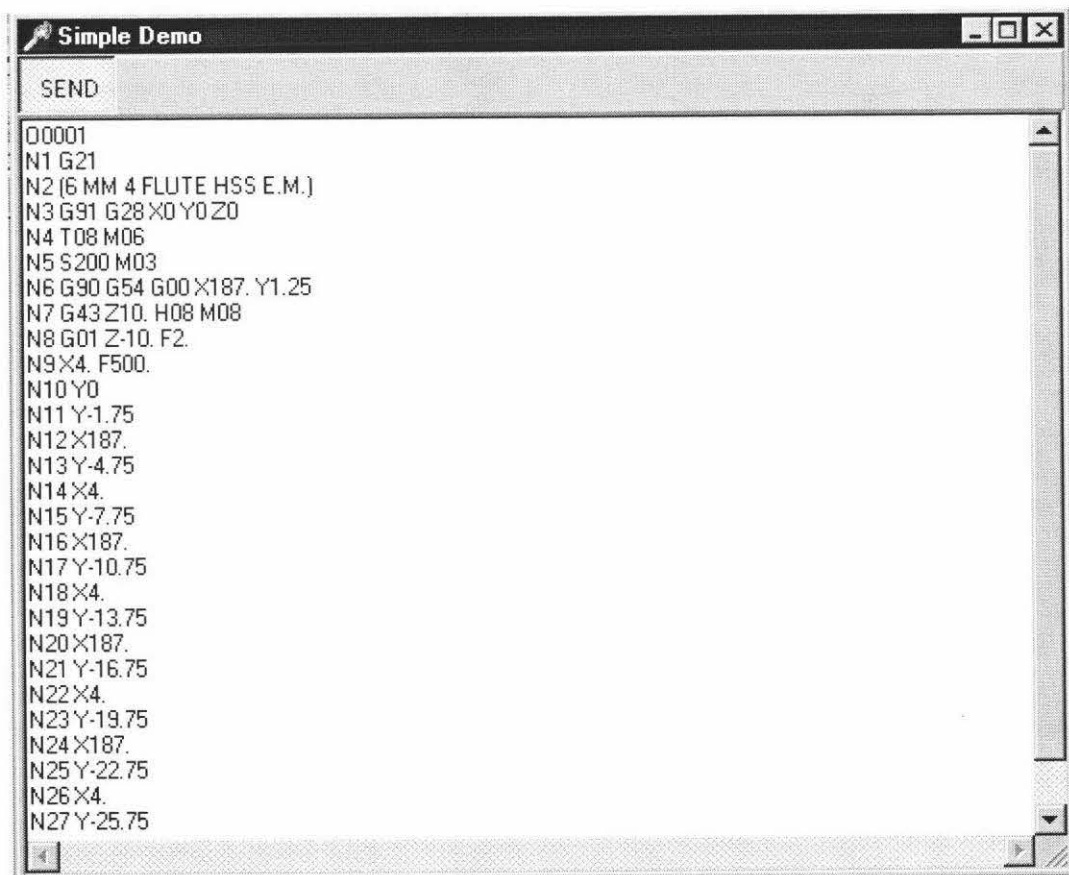


Figure 6.1: Character Transfer Interface

The Data Exchange interface (SimpleForm) has a Memo that is aligned in the centre of the interface, and has a sticky button on top which activates the read thread procedure required.

To access the character exchange interface, the user first needs to establish communication between the two terminals by configuring the COM ports on both terminals using the setting button on the menu. The port setting on the second terminal must also concur with the main workstation. By clicking on the quick button shown in Figure 6.2, the character exchange interface is then activated.

This interface allows the user on both end of the RS-232 network to type whatever character on the keyboard, and to have the characters displayed in the main memo.

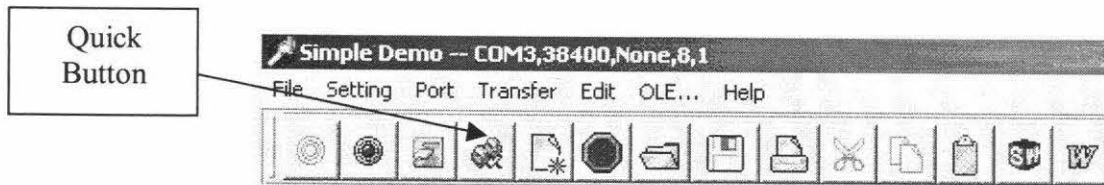


Figure 6.2: Quick button for Character exchange interface

6.3. Threading Applications

The Win32 operating system provides the user with the capability to have to have multiple threads of execution in the application developed in this project. Threads provide a mean for running many distinct code routines simultaneously. Of course, strictly speaking, two threads can't truly run simultaneous operations. However, each thread is scheduled fractions of seconds of time by the operating system in such a way as to give the feeling that many threads are running simultaneously.

6.3.1. The Thread Class

Delphi encapsulates the API thread object into an Object Pascal object called Tthread. Thread class is never used directly, because it is an abstract class- a class with virtual abstract method. To use threads, the user must always subclass Tthread and uses the features of this base class

The Tthread object has many properties and methods to handle threads. The properties and methods used are summarised with description of each in Table 6.1 and 6.2.

Table 6.1: Property Description

<i>Property Description</i>
<ul style="list-style-type: none"> • Free On Terminate - Determines whether the thread object is automatically • Priority - Specifies the thread's scheduling priority. Set this priority to a higher or lower value when needed. • Suspended - Specifies whether the thread is discontinued or not. • Terminated - Determines whether the thread is about to cease. • ThreadID - Determines the thread's identifier.

Table 6.2: Method Description

<i>Method Description</i>
<ul style="list-style-type: none"> • DoTerminate() Calls the OnTerminate event handler without terminating the thread. • Execute() Contains the code to be executed when the thread runs. • Resume() Resumes a suspended thread • Suspend() Pauses a running thread. • Terminate() Signals the thread to terminate.

6.3.2. ReadThread.pas

The most straightforward way to create Tthread descendants is to select Thread object from the New Item dialog box provided by Delphi. After choosing Thread object from the new item dialog box, the user will then be presented with a dialog box that prompts you to enter a name for the new object. In this case, enter TReadThread, this will then create a new unit that is initially defined as follows

```
type
  TReadThread = class(TThread)
  private
    { Private declarations }
  protected
    procedure Execute; override;
  end;
```

In this application, the ReadThread unit is used to display the character sent out from the other end terminal. Therefore, ShowData procedure had to be added to carry out this task. The procedure starts by declaring a variable lend that hold the length of the text within the form. This variable is then used to control buffer size limit (m_buf) if the length of variable lend exceeds the set limit.

```
if(lend>25000)then
begin
  { Edit Control buffer size limit }
  GhForm.Term.Text := string(m_buf);
  Exit;
end;
```

The method that the user must override in order to create a functional descendant of TReadThread is the Execute () method. In the listing shown in next page, Execute () will set GhExit to true to terminate the read thread before close the serial port.

The `sio_read` API function takes the COM port number, Buffer pointer, and the length of the data to be read as input argument and returns a long Integer variable `Len`. If the value of `Len` is greater than 10, the procedure then sets a null terminated string and then adds the synchronise method.

```

procedure TReadThread.Execute;
var
  len : LongInt;
begin
  (* before close port,set GhExit to true to terminate
  the read thread *)
  while not GhExit do
    begin
      Sleep(10);
      len := sio_read(GCommData.Port,@m_buf,511);
      if (len>0) then
        begin
          m_buf[len] := Char(0);{null terminated string}
          Synchronize(ShowData);
        end
      end;
    end;
  end;
end;

```

6.3.3. Advantage of Single Thread

The data exchange is a single threaded interface. As a single-threaded interface, it greatly reduces the complexity of the application. Win 32 requires that each thread creates a window have its own message loop using `GetMessage()` function. Therefore, when there are many threads being used, there will be messages coming into the application from a variety of sources. Because an application's message queue provides a means for serializing input-fully processing one condition before moving on to the next. Adding additional message effects the serialisation, thereby opening up potential synchronization problems and possibly introducing a need for complex synchronization code.

CHAPTER 7

7. File Transfer

7.1. Introduction

After a system has been properly set up and has the ability to exchange alphabetical character between terminals. The next milestone within the project is to be able to successfully transfer files. To achieve this task, the system requires the correct RS-232 cabling, correct serial port settings and both terminals involved in the transaction also need to be set up exactly the same. This chapter will cover areas such as file transfer protocols and the techniques used in the programs.

7.2. File Transfer Protocols

One of the main prerequisite before conducting a file transfer is to look for the protocols common to both machines and select the most appropriate one for the task. Following is a brief overview of some of the more well-known protocols.

XMODEM: The XMODEM protocol was one of the first file transfer methods that achieved widespread use on the desktop. XMODEM is a relatively simple protocol that allows a user to perform a binary transfer of a single file. It requires a clear 8-bit channel with no software handshaking.

Many minor variant of XMODEM have been developed over the years. The most universal is XMODEM-CRC, which uses a 16 bit CRC Checksum rather than an 8 bit additive checksum for improved error detection. XMODEM-1K increases the block size from 128 to 1024 bytes, giving great utilization of MNP-4, because it assumes an error free connection and does not require immediate acknowledgement of each packet.

YMODEM: is an enhancement of the XMODEM file transfer protocol. YMODEM adds a file information packet to the XMODEM protocol so that it can send the filename, size and data along with the file content. Because of this extra layer in the protocol, YMODEM can also send batch of files than just one file at a time.

ZMODEM: XMODEM and YMODEM work well under certain circumstances, but they have their drawbacks and limitations. The X & YMODEM work only on 8-bit communication lines. Packet-switched network cause XMODE performance to degrade, and most of the time X & Y MODEM don't make very efficient use of their available band width.

ZMODEM: was designed to correct all of these problems. First of all, ZMODEM was specifically designed to work well on packet-switched networks. This which allows to form a streaming protocol, meaning that it sends data in a continuous fashion without waiting for acknowledgment of individual blocks.

KERMIT: Kermit was developed in an attempt to let machines from various incompatible architecture communicate. Kermit is a carefully designed, well-layered protocol, with detailed specification and public domain source code available.

Kermit is a packet-oriented protocol that avoid using characters that could conflict with software handshaking or other protocol characters. It can work on either 8-bits or 7-bits channels or offer built in data compression and other advanced systems.

7.3. File Transfer Interface

This section describes much of the interface used in transferring data between terminals. As mentioned earlier, one of the main focuses of this application is to enable users to conduct quick file transfer between terminals. Therefore as shown in Figure 4.4, much of the file transfer configuration occupies half of the areas of the FtransForm (*FtransM.pas*) interface.

To initiate a file transfer through the application. The user first needs to establish communication through the serial port using the methodology and the steps illustrated in Chapter 5. Once the serial ports on both terminals have been activated and configured to the same standard, the Transfer menu button will be enabled and thus allowing the user to carry out the task of transferring files.

If the transferring menu button is clicked, the message window interface shown in Figure 7.1 will enquire the user to confirm the decision made in initiating a file transfer.

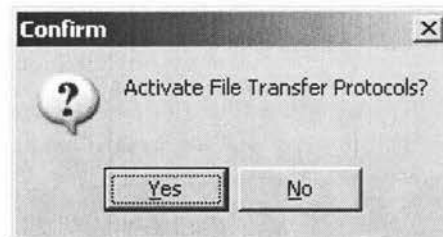


Figure 7.1: Confirmation Dialog

If the "Yes" button in the message dialog is pressed. The Function UpdateGT will be activated, which takes the user to the proceeding configuration panels to conduct file transfers.

7.3.1. Protocol Configuration Tab

After the user confirms to initiate the file transferring process. UpdateGT function only allows Tab sheet 1 to be made active. The UpdateGT provides a form of restriction that prevents unpredicted error, and it is also a method that can be easily to implement. Figure 7.2 illustrates Tab sheet 1, as it is shown it contains two radio groups, in which the user must specify which protocol the user wishes to employ, and whether a file is being transmitted or received.

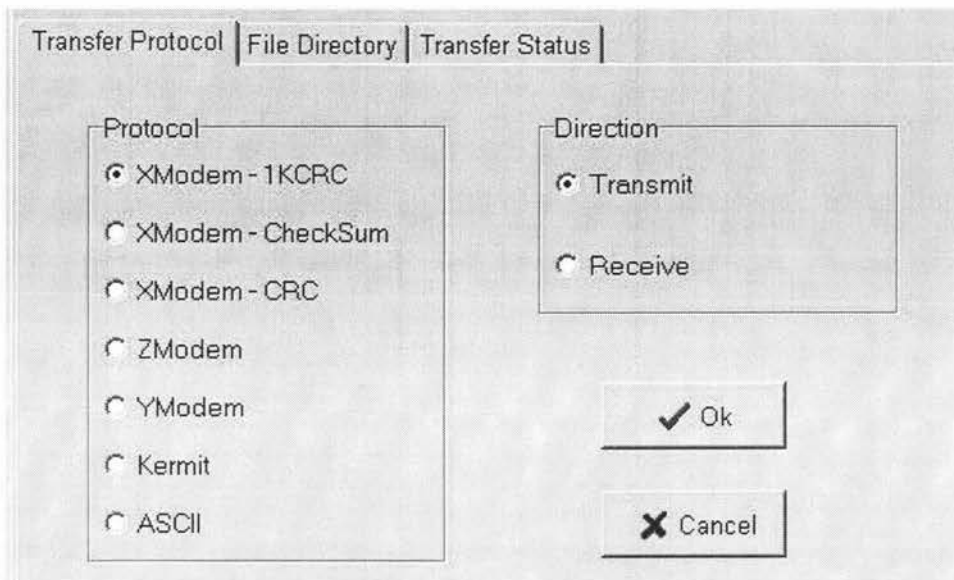


Figure 7.2: Protocol Specification Tab Sheet

Once the user selects OK, the items on the rgProtocol radiogroup will be passed to a shared variable (*Gprotocol*) that is declared in the single thread file (*FtProc*). The same applies to the rgDirection radiogroup found in TabSheet 1. The activation of the OK button on tab sheet also triggers a further update of the interface through UpdateHt function. This update allows TabSheet 1 to be continually active while enabling the user to access the functions in tab sheet 2.

7.3.2. Directory List Tab

As it is shown in Figure 7.3, TabSheet 2 acts like a directory dialog, and is composed of several directory navigating components.

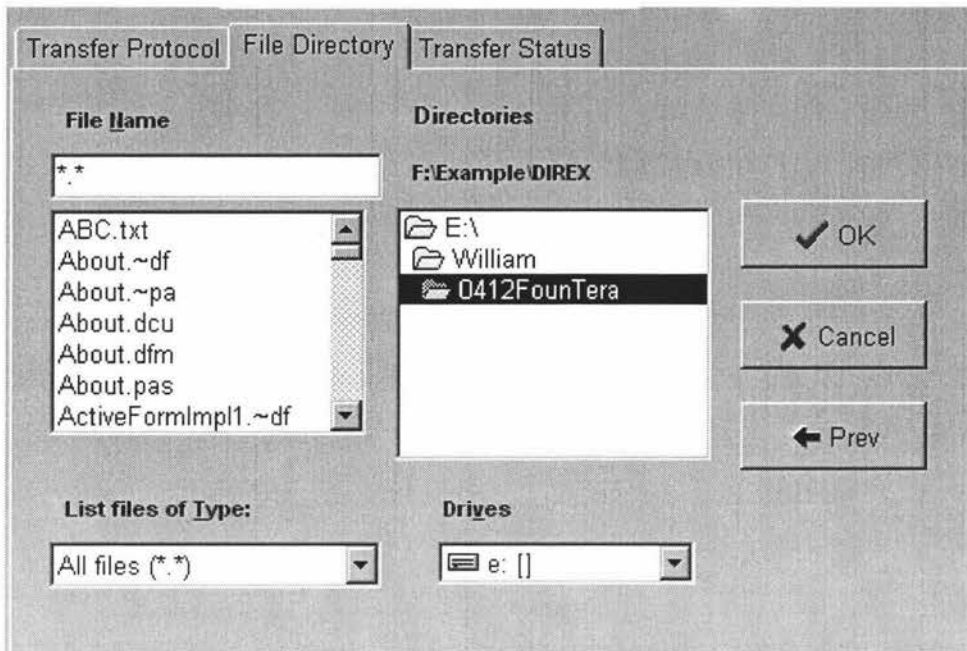


Figure 7.3 Directory List box

DirectoryListBox

The DirectoryList object displays a directory tree outline. User can double click FileListBox and use the keyboard to select directories in this window.

The user can use a DirectoryListBox alone in a window, but it usually needs to display files in selected directories. To do that, the user needs to assign the name of a FileListBox object to the DirectoryListBox's File List property. This could be carried out in the Object Inspector window. However, in this application, the assignment is made at a run time with code as listed below within the OnCreate event handler of the *FtransM* unit.

```
DirBox.FileList := FileListBox;
```

To show the currently selected path as a string, a label (DirLabel) was inserted into the form, and association was made at run time, by inserting the following statement into FtransForm's (*FtransM*) onCreate event handler:

```
DirBox.DirLabel := DirLabel;
```

DriveComboBox

A DriveCombox will allow the user fully utilise the storage spaces available in a PC, The following code was added to the FtransForm's OnCreate event handler to relate the DriveComboBox or the DirectoryListBox.

```
DriveBox.DirList := DirBox;
```

When users select a different drive, the DirectoryListBox automatically updates its tree. If the FileListBox is also associated with the DirectoryListBox, the file list is also updated.

FileListBox

A filelist box displays filename in the current directory. The DirectoryListBox is associated with a FileListBox within this application, so that the list automatically changes when user browses through directories.

The listing below shows the code required to add an edit control to this application, the EditControl initially shows the filter *.* , which selects all files. As users select filenames, the FileListBox inserts them into the Edit windows

```
FileListBox.FileEdit := FileNameEdit;
```

After the user has specified the path of the file to be transferred. The system will then record these specifications if the OK button (*DirDlgOKClick*) on Tab sheet 2 is clicked. The DirDlgOKClick first determines the direction of file transfer, by examining the ItemIndex on the rgdirection Radiogroup. ItemIndex holds the ordinal number of the selected radio button in the Items list.

The first button, Transmit (FT_XMIT) is 0, and the second button Receive (FT_RECV) is 1. The value of ItemIndex changes at runtime as the user selects radio buttons. If the user wants one of the buttons selected to appear, the user must assign that button to ItemIndex at design time; otherwise, leave ItemIndex will be set to the default value of -1, which means that no button is selected. The default itemIndex values in this case is set to 0, thus Transmit will be the default button chosen.

The listing below shows that if Transmit is the direction chosen, procedure *XmitFile* will be activated. *XmitFile* first declares a string variable (*Falcon*) that stores the FileNameEdit string. It then uses a Window's API function called *Istrcpy*, which copies the entire contents of one string (*Falcon*) into another string defined in *ActiveFormImpl1* unit (*GxFname*). The *XmitFile* function works closely with the *FTPProc* thread, which is the main thread used to execute the file transfers. (Details on *FTPProc* is discussed in the next section).

```
if FTTransForm.rgDirection.ItemIndex = FT_XMIT then
  XmitFile
```

If the direction of the file transfer is set as Receive, the *DirDlgOKClick* would then determine whether any the following three protocols ZMODEM, (FTZMDM), YMODEM (FTYMDM) or KERMIT (FTKERMIT) have been chosen in Tab Sheet 1. If so, the *DirDlgOKClick* procedure will again activate the *Istrcpy*, and copy the entire contents of one string (*Falcon*) into another string defined in *ActiveFormImpl1* unit (*GrPath*).

```
Istrcpy(GrPath,PChar(FTTransForm.DirBox.Directory));
SetCurrentDir(GrPath)
```


7.3.3. Transfer Status Feedback Tab

The DirDlgOKClick leads to the initiation of the third Tab Sheet, which is illustrated in Figure 7.4. This is also the last of the three tab sheets used with the file transfer sequence

The screenshot shows a dialog box with three tabs: "Transfer Protocol", "File Directory", and "Transfer Status". The "Transfer Status" tab is active. It contains the following fields:

- Port**: Input field with ID `lbPort`
- Protocol**: Input field with ID `lbProtocol`
- File Size**: Input field with ID `lbFSize`
- Length**: Input field with ID `lbLen`
- File Name**: Input field with ID `lbFname`

At the bottom center, there is a button labeled "X Cancel".

Figure 7.4: Feedback Tab Sheet

This interface provides the user a visual feedback of the current file transfer status through the *Ftproc* thread. Therefore, the detailed discussion on the technique in designing the feedback function will be described in the proceeding section.

7.3.4. FtProc Thread

As mentioned in the previous chapter, thread provides a very easy technique in delivering the necessary background processing while still providing the best possible response time. In this part of the application, the *FtProc* thread is used for numerous file transfer functions, and transfer status feedbacks.

As described in the preceding section, the file starts by declaring the ItemIndex value for the direction of file transfer Constants, and proceeds by declaring the ItemIndex value for the file transfer protocols. In both of these radio groups, the itemIndex holds the ordinal number of the selected radio button in the Items list (The first button is 0.). The value of ItemIndex changes at runtime as the user selects radio buttons.

7.3.5. Execute ()

After creating the thread object in main process, 'Execute()' procedure will be called automatically. The execute procedure starts by setting 'ret' as a long integer variable used by all the PComm functions. In this part of the FtProc, the 'Execute()' examines whether the rgdirection of the first radio group has been set to transmit. If the rgdirection is set to transmit, the program then determines the protocol chosen at tab sheet one, and applied the API accordingly. For example, if the user decides to conduct the file transmission through the ZMODEM protocol. The following API will be executed:

FTZMDM:

```
ret := sio_FtZmodemTx(port,GxFname,xCallBack, 27);
```

This API function takes on the following as input arguments: COM port number, the name of the file, and a call back function that is invoked each time data is transmitted and keeps the progress of the file transferring updated.

If the user opts to receive a file on Tab sheet 1, the Execute () will then undergo through a case statement that is similar to the one used for the FT_XMIT. The only difference is a different set of APIs are used for receiving. For example, if the user decides to receive file through the ZMODEM protocol. The following API will be executed:

```
ret := sio_FtZmodemRx(Port, fname,1,rCallBack, 27);
```

At the end of `Execute()`, the procedure lists the action that will be implemented when an error occurs during the transmission/receiving of the file. As shown in Appendix A, the procedure enters the `ProcessRet()` procedure when an error occurs (The `ret` is `<0`), and if the `ret` value checks out to be `>0`, the file would then display the message dialog box to give a warning signal

7.3.6. `xCallback/rCallback ()`

The Cancel button in Tabsheet 2 & 3 has been designed to a more complicated level than the Cancel button on tab sheet one. When a user pushes the cancel button on tab sheet 1, none of the setting in the radiogroups would have any effect, because most of arguments would have remained dormant, However, the Ok button on tab sheet one will invoke the `DirDlgOKClick ()`. Therefore, when the user wishes to exit form the file transfer sequence, the Cancel button executes another procedure, where the function calls the `xCallback ()` in *FtProc*.

When the cancel button on the second tab sheet is pushed, the application enters `DirDlgCancelClick ()`, and set the `GftCancel` to true. The `xCallback ()` from the *FtProc* will set the `xCallback` to -1, which would stop the terminal from transmitting data. The `PComm` file transfer API executed would then return `SIOFT_FUNC` (Described later in `ProcessRet` function), to cause the current file transmission to be aborted.

However, if none of the cancel buttons have been activated during the file transfer sequence. The `xCallback ()` would enter the `RefreshDlg()` on `FtransM` unit, and would update the caption edits on the tab sheet 3, and sets `xCallback ()` to 0 to indicate the application is continuing receiving file.

When receiving files, the application works exactly the same way. Except when `GftCancel` to true, the *FtProc* would enter the `rCallback()` function which operates exactly the same way.

7.3.7. ProcessRet()

As mentioned in the earlier section, the Execute () would enter the ProcessRet() when ret < 0. The ProcessRet() is specialised in returning message dialogs to give user feedbacks concerning the possible cause of failures.

The following table is the list of possible returns:

SIO_BADPORT	Port is not opened in advance
SIOFT_TIMEOUT	Protocol timeout
SIOFT_FOPEN	Can not open files
SIOFT_CANABORT	CAN signal abort
SIOFT_PROTOCOL	Protocol checking error abort
SIOFT_WIN32FAIL	Calling Win32 function failed

7.4. CNC Machine Quick Access

One of the key requirements in delivering a successful DNC system is the ability to transfer the data quickly and easily across the network. To achieve this task, the following outline components were added onto the main interface (*FtransForm*) as illustrated in Figure 7.5. This enables the users to add and link additional CNC machine or PC terminals to the main communication centre.

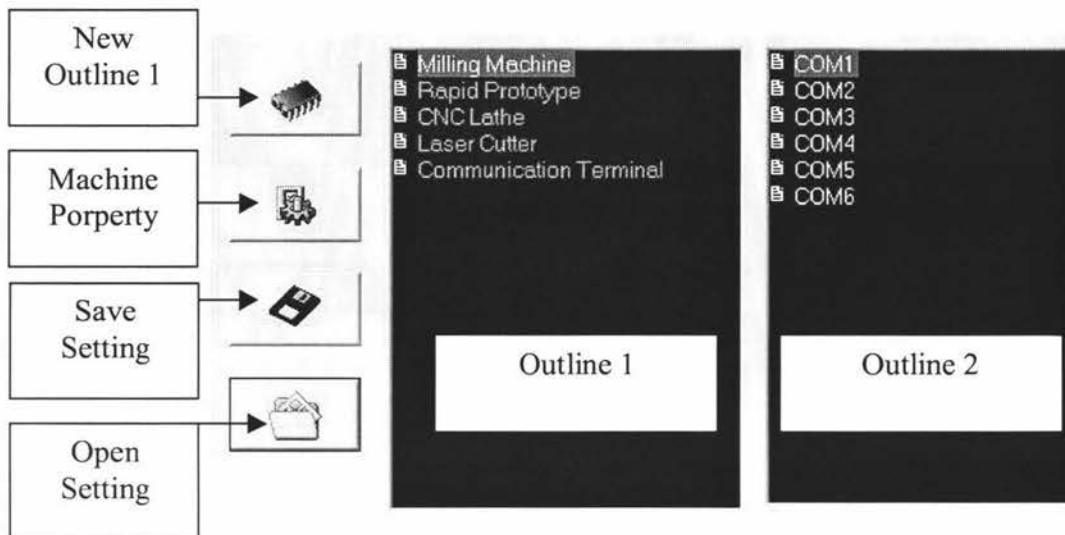


Figure 7.5: Quick File Transfer Interface

Machine outline contains some of the machine that a typical small-medium New Zealand CNC company might possess. The main function provided by these two outline component is that the user can add a new CNC machine quickly, and access it through one of the 6 serial port that are available on the main DNC workstation (2 standard serial port on a Pentium PC, and 4 additional serial port provided by the Moxa multiple port card).

The outline component on the main interface allows the users to assign CNC machine to any existing serial ports on the main workstation. Figure 7.6 shows the assignment of a milling machine to serial port 2.

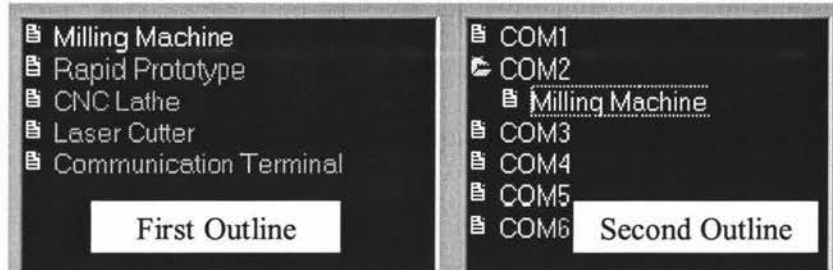


Figure 7.6: Machine assigned to serial port 2

The Handling the dragging method is used to allow the users to assign CNC machines to the serial ports. The method starts when the program calls the BeginDrag procedure after the user presses the left mouse button over the first outline component.

As soon as the button is released, the program automatically calls the EndDrag method of the first outline. The second outline defines a simple handler for the OnDragOver event, and performs the real work in the OnDragDrop event.

This method listed in the next page is quite complex, but it is also the main focus of this part of the interface. When the user drags a new element, the program first determines the item of the destination outline on which the element was dropped, using the GetItem function and the coordinates passed by the event. Then the program selects this item as the outline's current item -- that is, the item that will be affected by the following call to the AddChild method. The -1 is needed because the Lines array is zero-based, while the items are numbered starting from 1. It is possible to extract the text of the item directly from the Lines Array only because the items of the source list have no indentation.

```

procedure TFTransForm.Outline2DragDrop(Sender, Source: TObject; X,
Y: Integer);
var
    Current: Integer;
begin
    Current := Outline2.GetItem (X, Y);
    if Current > 0 then
        begin
            Outline2.AddChild (Current, Outline1.Lines[Outline1.SelectedItem - 1]);
            Outline2.Items [Current].Expanded := True;
        end
    end;
end;

```

The add button on the right hand side allows the user to add specific type of Date Terminal Equipment by typing in the name of the additional machine using the MachineForm shown below:

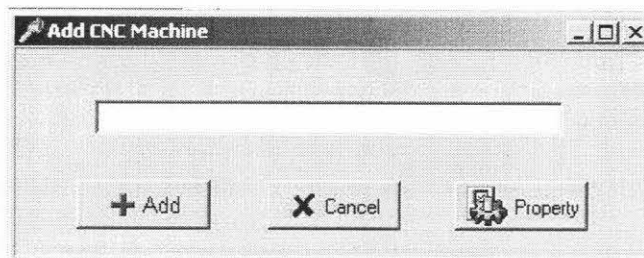


Figure 7.7: Machine Form

The following listing is used to examine whether the user have inserted a name for EditNew. If the EditNew is not empty or the IndexOf () is less than 0, then the method would then put the newly added machine name into the first outline.

```

if (EditNew.Text <> "") and
(FTransForm.OutLine1.Lines.IndexOf (EditNew.Text) < 0) then
    begin
        {add the string to both listboxes}
        FTransForm.OutLine1.Lines.Add (EditNew.Text);
        Close;
    end;
end;

```


The MachineForm has a button which links to the property form that is illustrated in Figure 7.8.

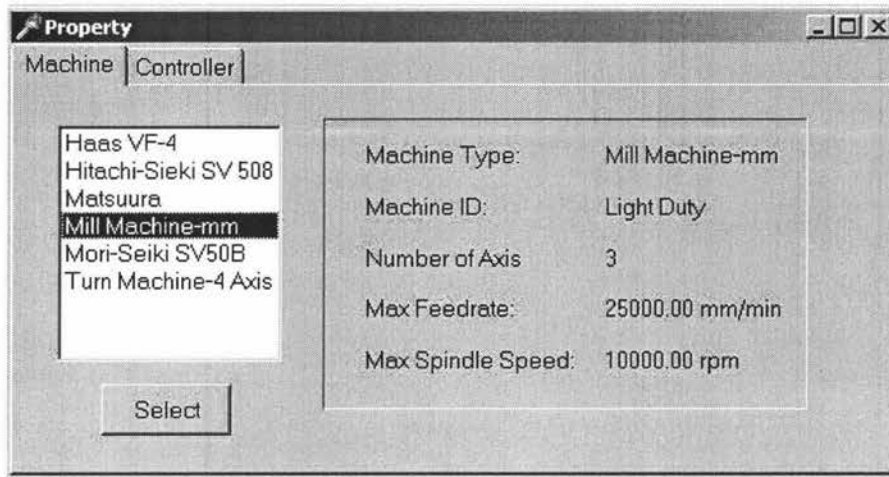


Figure 7.8 Property form

Through the property form, the application allows the users to specify the controller type through a for loop that assigns detailed description of each controller. The listing below shows the for loop assigns a value to every list items found in the listbox. When the ListBox1Click method is called, the for loop determines the value of the item selected, and displays the details in all the captions within the right hand side of the form.

```

procedure TPropertyForm.ListBox1Click(Sender: TObject);
var
  ListItem: Integer;
begin
  for ListItem := 0 to ListBox1.Items.Count - 1 do
    if ListBox1.Selected [ListItem] then
      begin
        if ListItem = 0 then
          begin
            MachTypeLabel.Caption := 'Haas VF-4 Mill';
            MachDutyLabel.Caption := 'Light Duty';
            NoAxisLabel.Caption := '4';
            MaxFeedLabel.Caption := '25000.00 mm/min';
            MaxSpinLabel.Caption := '10000.00 rpm';
          end;
        end;
      end;
  end;

```

CHAPTER 8

8. MDI Programming

8.1. Introduction

As part of the research, Multiple Document Interface (MDI) is used for both the NC code editor and the OLE windows (See Chapter 9). Every MDI has three basic parts

- The MDI Main window form
- One or More document MDI child-window forms
- The MDI main menu

Unlike in conventional Windows programming, a Delphi form object takes the place of the standard MDI frame and client windows. Classically, the frame window is the visible one; the client window is kind of silent partner that handles global operations, created child windows, and performs message services. In Delphi applications, the frame and client window still exist, but the user rarely uses them. For all practical purposes, it is possible for the programmers to treat the frame and the clients as one window, represented to the program as the main-window form. A MDI is often considered as a file-handling system, but an application's child windows do not have to be associated with disk files. The user could also use the MDI to construct multi window applications in other areas.

8.2. MDI- Main Window Form

In order to have MDI based interface, one of the first task carried out when creating the `FtransForm`, was that it must have its `FormStyle` set to `fsMDIForm` in the dialog shown in Figure 8.1. To ensure that the windows' form object is automatically created, the user could double check this by going to the `Project|Options`, and verifies that the `FtransForm` is shown in the main form list box and is listed under `Auto-created forms`

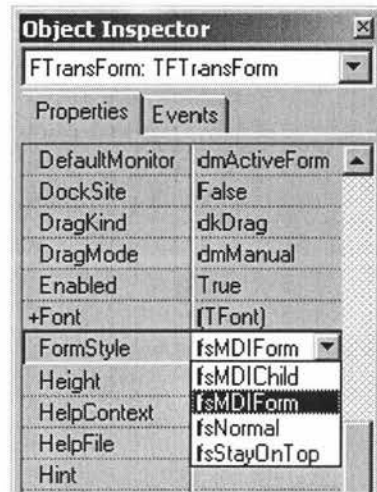


Figure 8.1: Changing the FormStyle

8.3. NC Code Editor

Every MDI application needs at least one child-window form and unit. In this project, there are 2 major functions that requires the use of MDIs. The first MDI application is the NC editor. With the NC editor, an NC program can be modified before it is transmitted to other terminals on the shop floor. The NC editor facilitates users to accomplish the task of modifying variables such as feed rate, spindle speed. Once the user has made the necessary modifications, the user can then download the NC programs to a specific machine tool immediately from the main communication terminal.

To make the NC editor into an MDI application. A new form object named ChildForm (*Child.pas*) must have its FormStyle set to fsMDIChild, and the form should be resized, so that it can be selected more easily.

Figure 8.2 illustrates the NC editor working with multiple documents like earlier versions of Microsoft Word. Many different programs can be opened at the same time as long as the RAM of the PC allows.

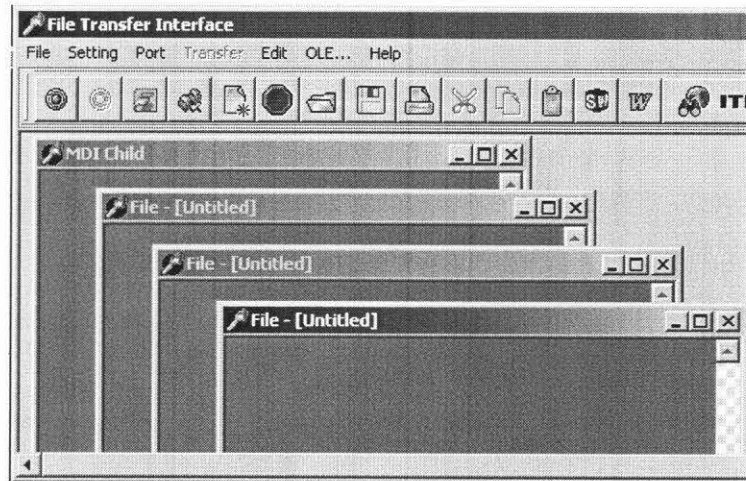


Figure 8.2: Multiple NC editor

Figure 8.3 shows an NC program in Editing. Once the modification is completed, the NC program can then be saved, downloaded, or uploaded to remote controller.

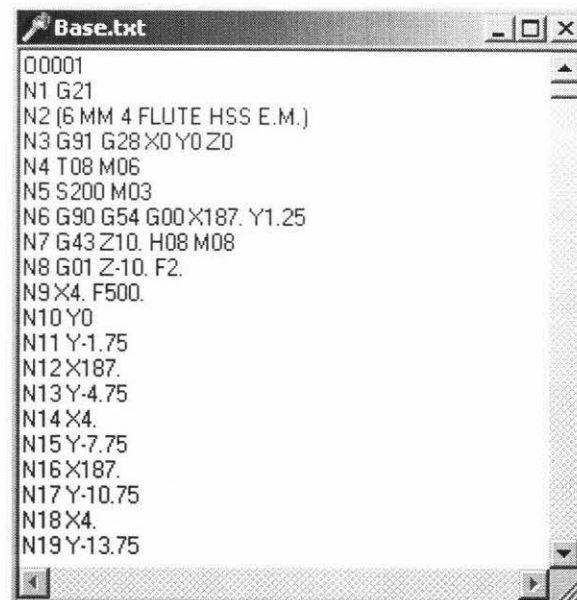


Figure 8.3: NC program in editing

8.4. *Child.pas*

The code in *Child.pas* is quite straightforward, this file interacts with numerous menu and speed buttons in *FtransForm* (*FtransM.pas*). The first related procedure found in this file is the *SetOLEFileName* procedure. This is the same naming procedure (*SetOLEFileName*) found in *Childwin.pas*, which will be describe later in the OLE MDI Section.

The next useful procedure that works in conjunction with the *FtransForm* is the *FormClose* procedure. This procedure exams whether the childform has been modified, and whether any text have been written on the memo. If these two conditions are met, the save procedure found in *FtransM* is then activated. The *Action := Cafree;* line frees the current childform, and the line found at the bottom of the *FormClose* procedure disables the speed button when it is not required

The rest of the procedures found in this file is used for controlling the status of the speed buttons. As illustrated in Figure 8.4, the *Memo1Click* procedure exams the *SelLength* to determine the length, in characters, of the selected text. This allows the application to have similar function found in Microsoft word, where cut and copy button will only be enabled when text have been selected.

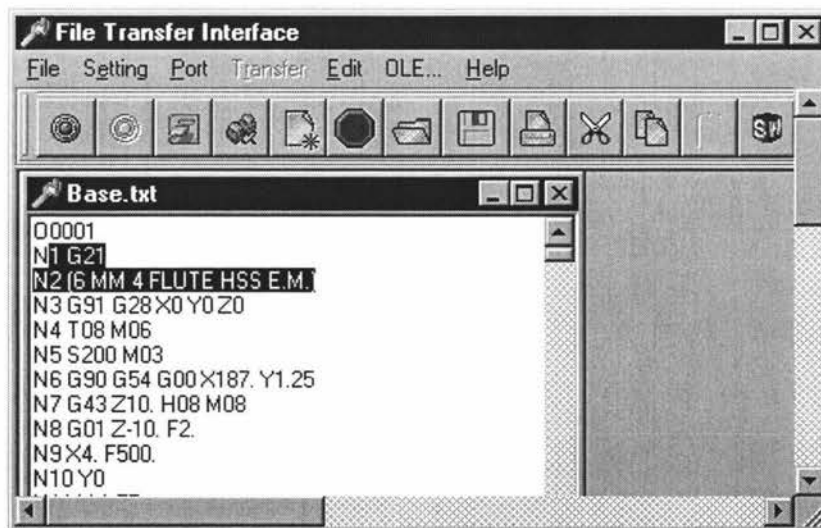


Figure 8.4: Cut & Copy button enabled when text is selected

8.4.1. FileNewClick Procedure

Many of the menu function items illustrated in chapter 5 are used to support the Multiple Document Interface. As shown in Figure 5.7, the first menu function found in *FtransM.pas* is the FileNewClick Procedure. Like most of the MDI Window based interface, this function creates a new Child Form for the NC editor. The structure of this procedure first checks whether the existing ChildForm has been modified, and whether the data has been saved. If the conditions are met, the procedure will make an addition increment to the number of ChildForm within the application.

8.4.2. FileOpenClick Procedure

The OpenClick Procedure will open up any NC codes within the NC editor ChildForm. In the procedure, the try block is used to define the code for which an exception might be raised. The try statement tells the compiler to try and get the Memo panel on ChildForm to open up files from the open dialog. If the code works, the except block is ignored and program execution continues. If any of the statement inside the try block raise an exception, the code within the except block is executed.

8.4.3. SaveAs Procedure

The procedure here examines whether there is any ChildForm within the application, and whether the save dialog have been activated. If both of these conditions are met, this procedure stores up the lines from the memo panel to a specific destination drive.

8.4.4. FileSaveClick Procedure

The save procedure is practically the same as the save as procedure. However, the save procedure will examine whether a filename has been assigned to the ActiveMDI child. If there is no name assigned, the fsave procedure activates the SaveAs procedure, if there is a name, then the procedure saves the data to the current name.

8.4.5. FilePrintSetupClick Procedure

Printing is an everyday necessity for most Windows users. Delphi provides the common print and print setup dialog boxes for use in the applications. The user in *FtransM.pas* can use the Print dialog box just before the printing begins and the Print set up dialog box to configure the printer.

The Print Dialog Box is encapsulated in VCL in the PrintDialog component. As with the other common dialog boxes. This component provides functions such as printer selection, number of copied, and orientation options are also available. The PrintDialog component has the execute Method only, and no events, thus it is very easy to implement. The following are the only event handler that is required.

```
procedure TFTransForm.FilePrintSetupClick(Sender: TObject);
begin
  PrinterSetupDialog.Execute;
end;
```

8.4.6. FilePrintClick Procedure

The key to this section is the use of the AssignPrn procedure, this connects a file with the printer. After starting the print process, the use can start using Write and WriteLn to print the text by calling the ReWrite Procedure. Using a for loop from the first to the last line

```
for I:=0 to ChildForm.Memo1.Lines.Count-1 do
  WriteLn(PrintFile, ChildForm.Memo1.Lines[I]);
```

In Delphi, a try block can be followed by either an except or finally block. In This procedure a try block and a finally block are used to perform the clean up action after the file in the memo panel has been printed.

CHAPTER 9

9. COM & OLE Automation

9.1. Introduction

Although this chapter is very much concerned with the concept of OLE (Object Linking Embedding) automation technology. However, it is also the purpose of this chapter to discuss the concept of COM (Component Object Model) and how it relates to the later concepts such as OLE and ActiveX.

9.2. Component Object Model (COM)

The Component Object Model (COM) forms the foundation upon which OLE and ActiveX technology is built. COM defines an API and a binary standard for communication between objects that is independent of any particular programming language or platform. COM objects are similar to the VCL objects, except that COM objects have only methods and properties associated, not data fields.

9.2.1. Interfaces

A COM object consists of one or more interfaces, which are essentially tables of functions associated with that object. COM defines a standard map of how an object's function is laid out in memory. Functions are arranged in virtual tables. The programming language description of each table is referred to as an Interface.

An Interface can be divided into 2 parts. The first part is the interface definition, which consists of a collection of one or more function declaration in a specific order. The interface definition is shared between the object and the user of the object. The second part is the interface implementation, which is the actual implementation of the functions described in the interface declaration.

9.2.2. IUnknown

Just as all Object Pascal classes implicitly descend from Tobject, all COM interfaces implicitly derive from IUnknown. IUnknown is defined in the system unit as follows:

```
type
    IUnknown = interface
        [{00000000-0000-0000-C000-000000000046}]
        function QueryInterface(const IID: TGUID; TGUID; out Obj): Integer;
        stdll;
        function _AddRef: Integer; stdcall;
        function _Release: Integer; stdcall;
    end;
```

Apart from the use of the Interface keyword, another obvious difference between an interface and class declaration is that the preceding listing is the presence of Globally Unique Identifier or GUID.

9.2.3. GUID

GUID is an ID which identifies any COM server class and any interface in the system. GUIDs are created by a special COM library function called CoCreateGUID. This function generates a GUID that is guaranteed to be unique. CoCreateGUID uses a combination of the PC information, random number generation, and a time stamp to create GUIDs.

In Delphi, GUIDs are generated automatically when the user creates an automation object, COM Object, ActiveX Control, or ActiveForm Control. GUIDs in Delphi are defined by the TGUID record.

9.3. *Object Linking Embedding (OLE)*

As part of the development based on COM, the most sophisticated data-sharing technique available in Windows is known as Object Linking and Embedding (OLE). The key advantage that OLE offers is a shift from an application-oriented view of computing to one that centres on documents. With OLE, users can also combine information in unforeseen ways. For example, a word processor document can contain graphical image created by software unknown to the word processor's author. OLE makes it possible for users, not just software designers, to create new type of documents that are not limited to a single application specific format.

9.4. *OLE Object*

The application developed within this research project has 3 OLE objects from the registered Windows registry. The links are Microsoft Word, Microsoft Internet Explorer and SolidWorks 2000.

To create an OLE object, the application must first call the `CreateOleObject` function in Delphi's `ComObj` unit. This function calls a number of internal system wide OLE functions. The end result of these series of calls is that the function returns a COM object to the user containing an interface to the object that the user wants to call. As illustrated in the list below (FtransM), the call retrieves the Word application.

```
S := CreateOleObject('Word.Application');  
S.Visible := True;
```

The first line of the code `S:= CreateOleObject('Word.Application')` asks for an object called `Application` that resides inside `Word`. `CreateOleObject` retrieves an instance of the object in the form of an `Idispatch` interface encapsulated inside a `Variant` called `S`. Thus, the user can access the `Visible` property of the object by simply writing

```
S.Visible := True;
```

The call to `CreateOleObject` returns COM object called `Idispatch` housed inside a `Variant`. The user can pass a string to `CreateOleObject` specifying the name of the COM object the user want to retrieve. In this application the main Word Automation object was retrieved by passing in the string `Word.Application`. To find this string, it was required to work with the GUID Registry by opening the `REGEDIT.EXE` within Windows (See figure 9.1). This is a simple database that has the primary take out associating numerical values with each of the COM Objects available on the system. By opening the tree called `HKEY_CLASSES_ROOT`, it is then possible to select the string name required by scrolling down to the `Class ID (CLSID)` folder.

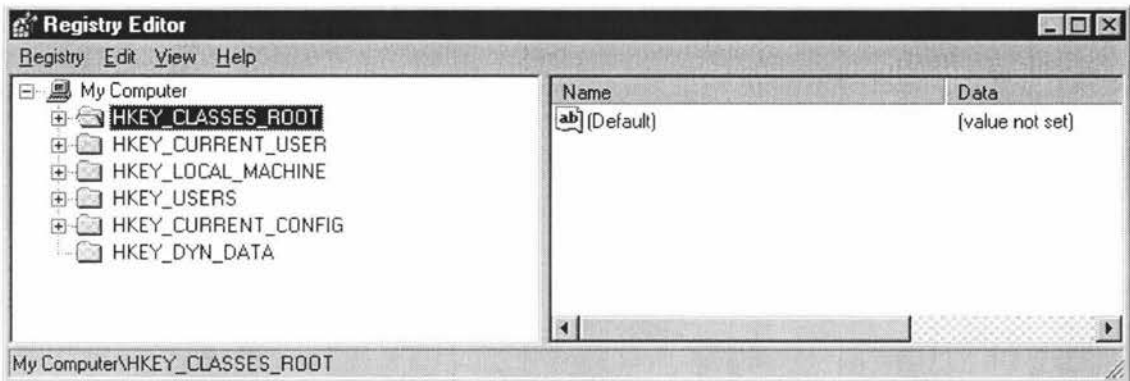


Figure 9.1: *RegEdit.exe*

9.5. MDI OLE Container Interface

The preceding sections demonstrate only one way to use OLE. Another way is to create a container application that communicates with an OLE server. With this technique, it is possible to link and embed server documents in the application developed within this research project. For example, an OLE container can create, load, edit and save a Microsoft Access document. Because Access is a full-featured server application, Delphi program can call on Access to create database documents, all from inside of the application window developed within this project.

To make a MDI child for the OLE container, similar techniques were outsourced from the MDI NC editor. When the user activates the “Activate OLE” from the menu on FtransForm. An OLE childform is created, and it calls the object’s InsertObjectDialog method, which displays the dialog shown in Figure 9.2.

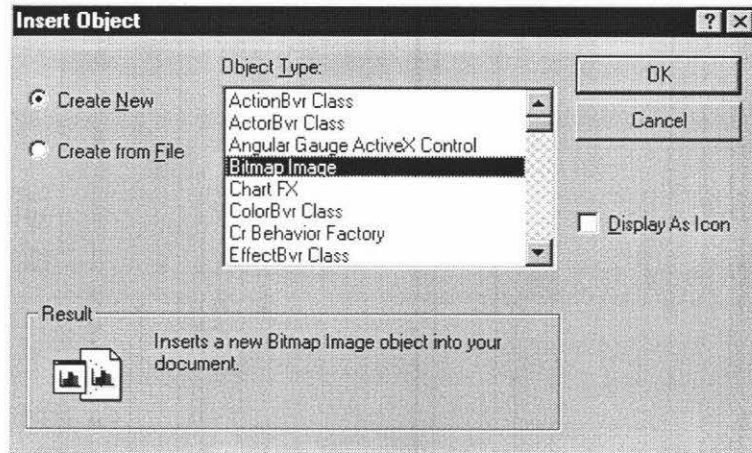


Figure 9.2: Ole Container

The user can select the type of OLE object to create and choose whether to insert a full image of the object or display it as an icon. Users can also select an icon to depict the file. Figure 9.3 shows the *Childwin.pas* containing window’s paint utility. The OLE server replaces the Delphi Application’s menu and displays with its own. With only a little programming, the sample program provides full graphic file editing capabilities to the users.

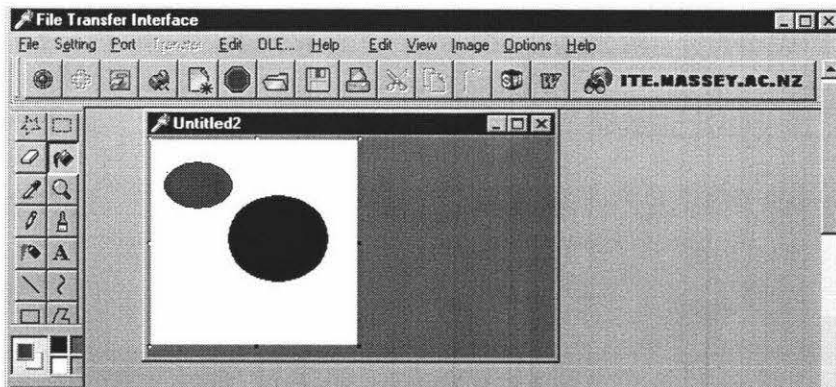


Figure 9.3: OLE Container in FtransForm

9.5.1. ChildWin.pas

The code *ChildWin.pas* is very much the simplified version of *Child.pas* used for the MDI NC editor, this file also interacts with OLE menu in *FtransForm* (*FtransM.pas*). As mentioned in the previous section, both *ChildWin* and *Child* units uses the same naming procedure (*SetOLEFileName*). This procedure allows file name to be extracted on to the top of the window on the *Child* form.

In the *SetOLEFileName* procedure, the property from the OO Pascal syntax was implemented. A property is basically a name that is mapped to some read and write methods or that accesses some data directly. In other words, every time that a user reads the value of a property or changes it, the user might be accessing a field (even a private one) or might be calling a method. In the case of *Child.pas*, the definition of a property for a data object is :

```
property OLEFileName: string read FOLEFileName write SetOLEFileName;
```

To access the value of the *OLEFileName*, this code has to read the value of the private field *FOLEFileName*, while to change the value it calls the method *SetOLEFileName*.

```
procedure TChildForm.SetOLEFileName(const Value: string);
begin
  if Value <> FOLEFileName then
  begin
    FOLEFileName := Value;
    Caption := ExtractFileName(FOLEFileName);
  end;
end;
```

CHAPTER 10

10. Remote Access using Active X

10.1. Introduction

As listed in the system specifications, one of the interfaces will provide remote access to functions developed within this research project. To meet this requirement, the research project aims to apply COM related technology, such as ActiveX to implement of remote access capability for the system through the World Wide Web (WWW).

10.2. Understanding ActiveX

ActiveX is a relatively new term for a technology that has been around for awhile originally ActiveX was called OCX controls. An ActiveX control are DLL-based. This means that when it is applied, the designer needs to distribute their code (the OCX file) along with the application.

An Active X control is essentially a COM object in disguise. The primary difference between An ActiveX control and a COM object is that an ActiveX Control has a design time interface. An ActiveX controls also has code that enables it to be deployed on a Web page or over a network. ActiveX is a subset of COM, everything discussed concerning COM object in the previous chapter applies to ActiveX control as well.

As a Delphi based research project, the components here has the capabilities of native VCL components and forms. However, by converting VCL controls into ActiveX controls, the potential market is not merely fellow Delphi and C++ builder developers, but also users of practically any Win32 development tools. Even if the end user is not a component vendor, the user can still take advantage of ActiveX controls to add contents and functionality to World Wide Web pages.

10.2.1. Installing ActiveX Component

Microsoft Internet Explorer version 3.x and above are based on ActiveX control, The user can import these ActiveX control into Delphi and use the control within the application.

To install the ActiveX control component, the user first needs to choose Component | Import ActiveX control from the main menu. Delphi would then pop up with the Import ActiveX illustrated in Figure 10.1.

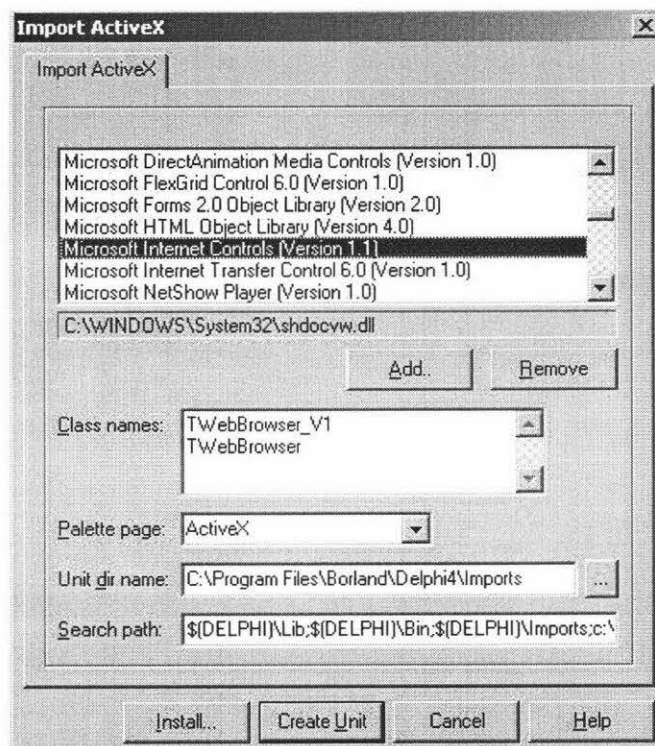


Figure 10.1: Import ActiveX Control

The user then needs to scroll down and select *Microsoft Internet Control*, and install it. After the control has been placed on the Component Palette, the class name such as *TwebBrowser* could then be used within the application.

10.2.2. ActiveForm Wizard

The process involved in creating an ActiveX form is very straightforward. By clicking the ActiveX form icon from the one-step wizard, Delphi will invoke the ActiveForm Wizard, which is shown in Figure 10.2.

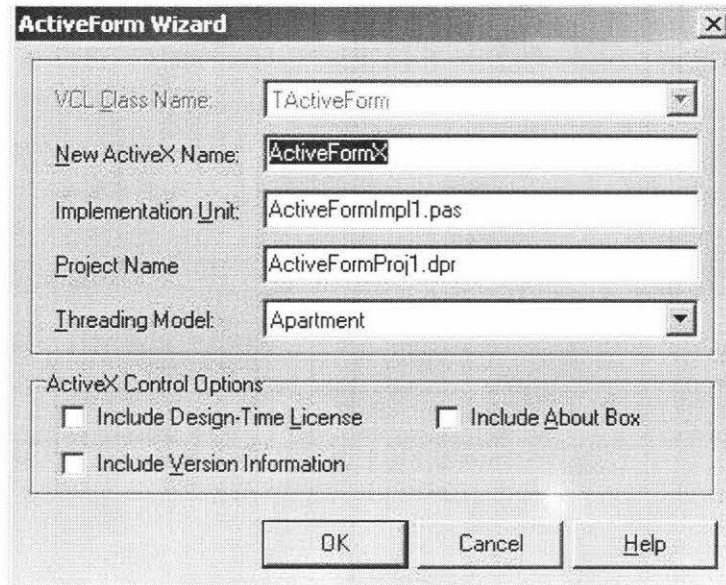


Figure 10.2: ActiveForm Wizard

10.2.3. Type Library

The ActiveForm wizard is a powerful tool provided by Delphi. When it is activated, it would also automatically generate a type library (Shown in Figure 10.3) that allows the user to add or remove interfaces, add properties and methods to interfaces, remove elements from interfaces and create host of other COM elements such enumeration, records, or co-classes.

As illustrated in Figure 10.3, on the left side of the Type Library Editor is the Object panel. The Object pane contains a tree view control. On top of the tree view hierarchy is the type library itself. Below the type library are elements contained in the type library.

On the right side of the Type Library Editor is the Information panel. This pane provides information about the object currently selected in the Object panel. The information presented in the Information pane varies with the type of object selected. The attributes page shows the type library name, its GUID, version, help string, help file, and so on.

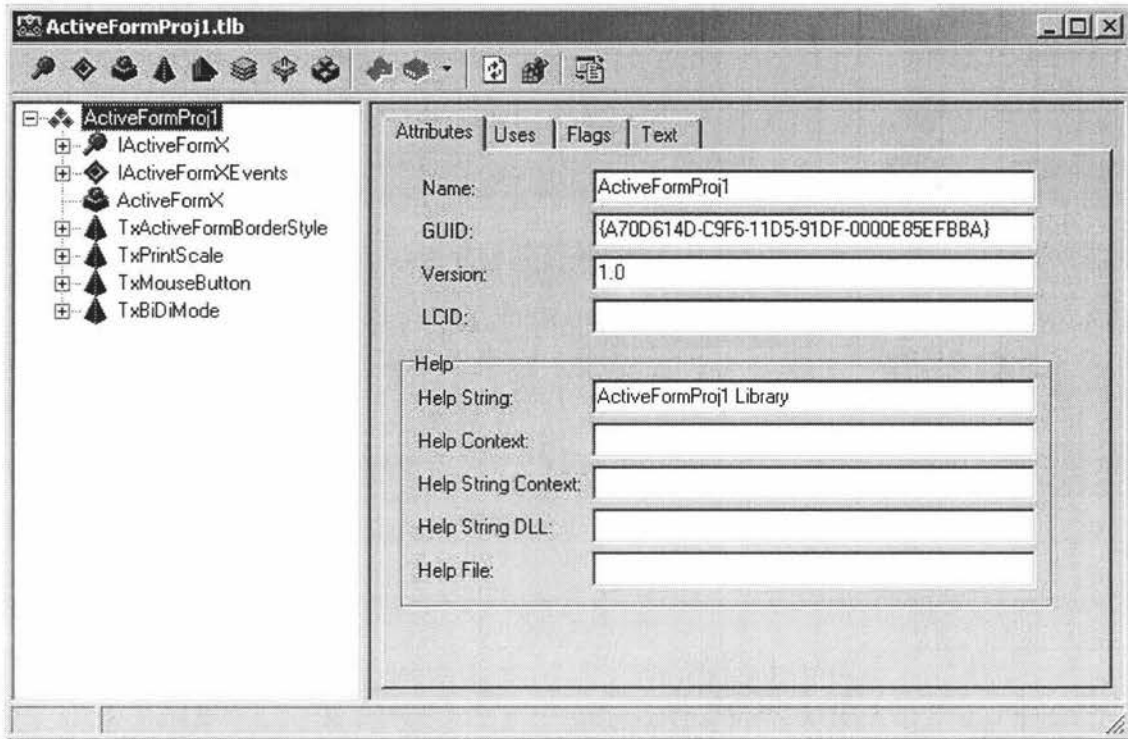


Figure 10.3: Type Library

When the type library node is selected, the Information panel shows a tab label uses. In almost all cases, this list will include the OLE automation library, it can also include others as well.

The text page shows the type library definition in IDL syntax. IDL for the ActiveFormProj1 is illustrated in Appendix A. It is a sort of scripting language used to create binary type library files

10.3. Building the Form

An ActiveForm form is just a regular form at this stage. The user can add controls to the form, add code, and respond to events just like a form that belongs to an application. One difference is that the title bar on an ActiveForm does not appear on the control itself. It is just there at design time.

The design of the ActiveForm uses several functions found in the FtransForm (*FtransM.pas*). Figure 10.4 shows the components that is found in ActiveForm.

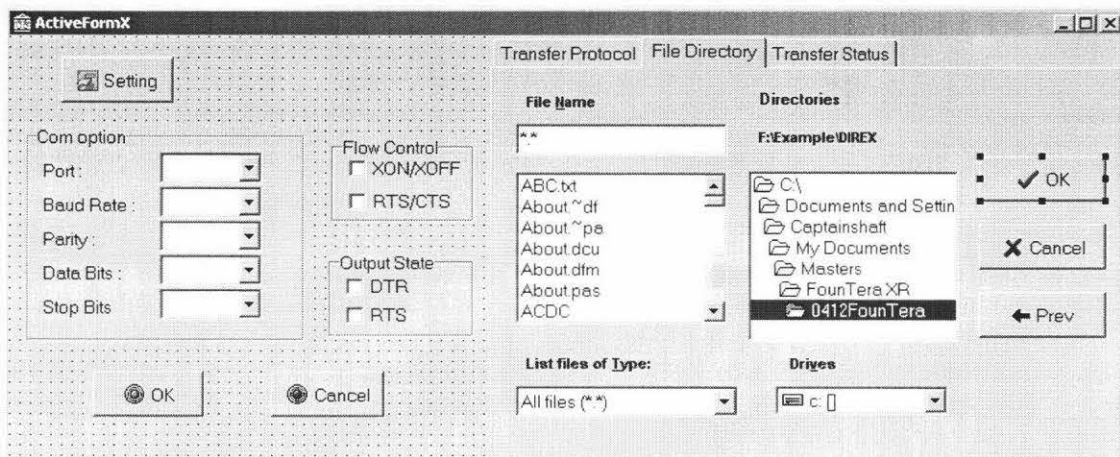


Figure 10.4 Build the ActiveForm in Delphi

The initial status of several button remains dormant until the ActiveForm calls the *cmSettingClick* method by clicking on the “setting” button. This method assigns the default data from *Exglobal.pas* into the ItemIndex for all the combo boxes found on the left hand side of the ActiveForm. The user must then press “OK” on the left hand side to access the file transfer sequence on the right hand side of this interface. The file transfer sequences used here have similar structure to the ones found in FtransForm.

10.4. Deploying an ActiveForm

Once the form have been built, the next step is to deploy the ActiveForm. In this case, the goal is to have it appear inside the Internet Explorer.

Start by choosing Web Deployment options from the project menu. A dialog like the one shown in Figure 10.5 will pop up. Before this dialog appears, the project must be compiled and linked, so a short delay is expected while the files are processed.

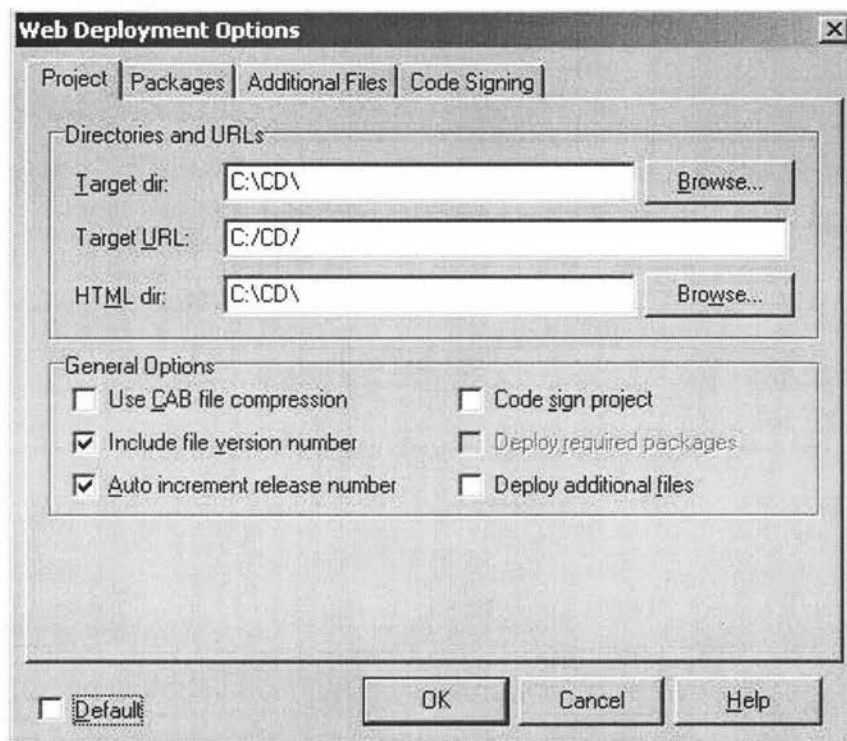


Figure 10.5: Web Deployment Options

At the top of the Deployment option dialog are three controls

- Target Dir
- Target URL
- HTML Dir

In the target Dir Field, the user needs to list where OCX or any other binary files will be deployed. These files can be distributed to anyone who attaches to the main server over the web. As the ActiveForm developed within this project was not saved into a Web server. The files were stored in a randomly created folder.

The target URL specify is used by the HTML an/or INF file that launches the OCX. The string enter in this field should point to the directory where the OCX is located when it is ready to be deployed.

By default, Delphi will cerate sample HTML and INF files for the project. The HTML file can be loaded into a browser and used to launch the OCX created. If the project deploys multiple files, the HTML file will reference a second file with an .inf extension. The INF file will contain the URL where the OCX resides, and any other additional files needed by the project, such as packages or the runtime library. In the case of *ActiveFormProj1* the runtime library are not used, so no INF file will be created.

HTML Dir indicates where the sample HTML and INF files that Delphi generates will be placed. Typically, this location is the same directory as our specified Target Dir.

Once all three controls have been specified, the user must make sure Auto Increment Release Number check box is checked. Then by choosing the Web deploy from the project menu in Delphi, the entire AvtiveForm project is copied automatically into the directories specified in the web deployment option dialog.

Once, the web pages is created, the user could test the interface by going to the HTML folder where the files have been deployed. In certain cases, the user may spot odd mistakes with the code, and wishes to see some changes being made to it. Although, the process of redeploying the entire project may seems to be the obvious thing to do, but when an OCX is loaded into the memory through Microsoft system, the only way it is unloaded is to reboot Windows every time an user wishes to make a change. In layman's term, when the user tries to redeploy the OCX, it might be getting the same OCX in the client app because the old DLL may not have been unloaded from memory. Furthermore, the OCX that is downloaded onto a machine are often stored in a directory called OCCACHE, which is just below the Window's *Downloaded Program Files*. Hence, to make the desired changes to the application, the user must unregistered and delete the files from this directory to create a clean machine to run tests on.

10.4.1. Connecting to an ActiveForm

At this point the project's OCX is ready downloaded to the second machine, and to access it through the internet. The end user can view the HTML file on the browser. To understand how this procedure works, consider the HTML generated by Delphi:

```
<HTML>
```

```
<H1> Delphi 4 ActiveX Test Page </H1><p>
```

You should see your Delphi 4 forms or controls embedded in the form below.

```
<HR><center><P>
```

```
<OBJECT
```

```
    classid="clsid:A70D6152-C9F6-11D5-91DF-0000E85EFBBA"
```

```
    codebase="C:/CD/ActiveFormProj1.dll"#version=1,0,29,0
```

```
    width=847
```

```
    height=321
```

```
    align=center
```

```
    hspace=0
```

```
    vspace=0
```

```
>
```

```
</OBJECT>
```

```
</center></HTML>
```


The CLSID shown here specifies the GUID associated with the object created. The line labelled codebase points to the directory where the OCX resides. Because the application is stored in a single machine that does not feature a web server, the target URL field in the Web deployment option dialog will display a DOS path rather than a URL

```
codebase="C:/CD/ActiveFormProj1.dll"#version=1,0,29,0
```

Figure 10.6, demonstrates that a every day web page could access the ActiveForm application, a free internet based web site has been created, and when activated the link will take the user to the ActiveFormProj1 site.

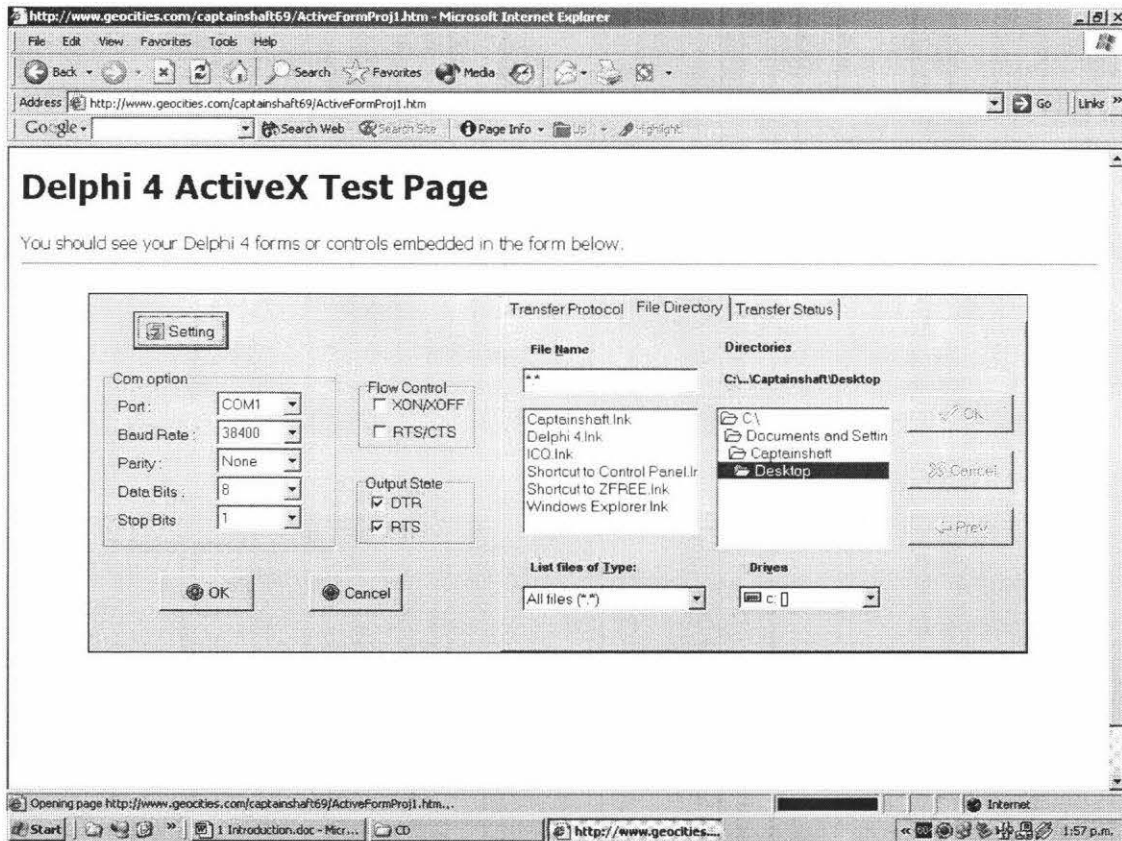


Figure 10.6: ActiveForm on the WWW

CHAPTER 11

11. Testing

11.1. Introduction

This section gives an overview of the testing procedures developed and implemented in the presented system. Few specific tests have been made to evaluate the system and the functionalities.

The testing for interface defects is particularly difficult because interface faults may only manifest themselves under unusual conditions, and because of the tight time restrictions on the project. The testing phase of this project has been merged with the implementation phase to a great deal.

11.2. Interface Testing

The interface testing used in this application takes place when modules or sub-systems are integrated to create the larger system. Therefore tests have been carried out on modules as they have been built. Each module or sub-system has a defined interface which is called by other program components. The objective of these testing is to detect faults which may have been introduced because of interface errors or invalid assumptions about interface.

The first interface testing is to examine each call to an component. This include designing a set of tests where all the values ranges available on the combo boxes are tested. For example, the *config.pas* has wide range of baud rate available for file transfer. To test whether these values could be used, file transfers were conducted at both low and high baud rate settings

The second level of testing is the application of stress testing. Some classes of system are designed to handle a specific load. For example, by opening numerous number of MDI child forms in the application and observe whether there are any circumstance that may arise through an unexpected combination of events where the load placed on the system exceeds the maximum design load of the system. In these circumstance, it is important that system failure will not cause any system corruption, or unexpected OS crashes. This technique is quite important, since the application is most likely to be used within an industrial environment, where down time in a DNC system may cause tremendous losses.

Another third testing consists of transferring data across different terminals. For example, PC or CNC machines. This will provide immediate feedback to see whether data have been successfully transferred.

11.3. Test Results

The results of the module tests conducted were only briefly looked at, as these were evident by the way the system works with low number of bugs.

The test at code level were successful, and the bugs that were found were fixed and checked again. The application also did well under the stress test, overall speaking the system performed well in dealing with many MDI child forms. However, the stress testing was carried out on a fairly recent machine. Hence, it had much more tolerance than some of the PC workstation found in small to medium manufacturing firms.

CHAPTER 12

12. Results & Discussions

The research project has proven to be a successful project. The application developed contains the following major components which fulfilled the criteria set out by the specification.

- *Character Exchange Interface*

This allowed the users to establish communication between the terminals after a serial connection has been made.

- *File Transfer Interface*

The file transfer made use of numerous serial transfer protocols, and enabled the user to use a easy step by step file transfer through the main interface within of the application.

- *NC Editor Child Form*

The NC editor enabled the user to alter the code before transmitting any file to a terminals. The NC editor was also very convenient for comparing two NC programs if the use wishes.

- *OLE Automation Child Forms & Buttons*

The OLE automation containers and buttons allowed other programs to be integrated into the application, and thus satisfied the objective of creating a CAD/CAM link to integrate manufacturing information flows between the users and the NC machines on the shop floor.

- *Internet Remote Access using ActiveForm*

The application of ActiveForm provided remote access through the World Wide Web (WWW). The powerful tool will allow users all over world to share/purchase the application developed within this research project.

CHAPTER 13

13. Recommendations

In order to further improve the application software, the following areas of development are recommended:

Database Design

The property form within the application software shows the fundamental concepts required in assigning controller type from a database to newly added machine type on to the outline component. Database design is an important feature within a DNC system, because any communication occurring between two entities involves visiting a related database or storing data into a related database. Although it was not specified in the application software specifications, but a fully developed CIM strategy requires a strong database system that liase between the DNC system and the shop floor manufacturing configuration. Further efforts must be made to identify what information should be managed for the local database, and what information should be exchanged with remote site databases via the COM automation technique.

Monitoring System

Another aspect of CIM is to have gain advance process control over the entire automated processes in one application software. Process monitoring requires further involvement of auxiliary hardware integration, such as PLCs temperature sensors, limit switches and motor controllers. The information collected by the monitoring system will be stored in the local database, or passed back to PLC for automated feedback control.

CHAPTER 14

14. Conclusion

This research project has successfully developed a working GUI that is implemented with the Moxa multiport card. The application software allows data to be transferred to and from other terminals. The software made use of the OLE technology that allowed access to other software installed on the desktop workstation, and ActiveX technology that allowed remote access to the file/data transfer components through the World Wide Web. The future of this field will depend much on the total integration of automated NC equipment and Information Technology, such as Internet and database.

15. REFERENCES

1. Calvert, Charles, 1999, *Charlie Calvert's Delphi 4 unleashed*, [Indianapolis, IN] : Sams Pub. : Borland Press.
2. Sholz-Reiter, B, 1992, *CIM interfaces : concepts, standards and problems of interfaces in computer integrated manufacturing*, London ; New York : Chapman & Hall.
3. Waldner, Jean-Baptiste, 1992, *CIM, principles of computer-integrated manufacturing*, Chichester, West Sussex, England ; New York : Wiley.
4. Smith, Graham, T, 1993, *CNC machining technology*, London ; New York : Springer-Verlag.
5. Swan, Tom, 1998, *Delphi 4 bible*, Foster City, CA : IDG Books Worldwide.
6. Pacheco, Xavier, 1998, *Delphi 4 developer's guide*, Indianapolis, Ind : Sams.
7. Dorf , C, R, 1994, *Handbook of design, manufacturing, and automation*, New York : Wiley.
8. Weck, Manfred, 1984, *Handbook of machine tools*, Chichester [West Sussex] ; New York : Wiley.
9. Canto, Marco, 1997, *Mastering Delphi 3*, San Francisco : Sybex.
10. Reisdorph, Kent, 1998, *Sams teach yourself Borland Delphi 4 in 21 days*, Indianapolis, Ind. : Sams Pub.

11. Lee, Geoff, 1993, *Object-oriented GUI application development*, Englewood Cliffs, N.J. : PTR Prentice Hall.
12. Adiga, S, 1992, *Object-oriented software for manufacturing systems*, London ; New York : Chapman & Hall
13. Nelson, Mark, 2000, *Serial communications developer's guide*, Foster City, CA : IDG Books Worldwide.
14. Moxa Technologies Co. Ltd, 1999, *Smartio C104H/HS User's Manual*, Moxa technologies Co, Ltd.
15. Sommerville, Ian, 1996, *Software engineering*, Wokingham, England ; Reading, Mass. : Addison-Wesley Pub. Co.
16. Lin, Edward, 1995, *Virtual Manufacturing User Workgroup*, Lawrence Associated Inc.
17. <http://www.ctips.com/rs232.html>
18. <http://www.moxa.com/product/PComm/pcomm.htm>
19. <http://www.moxa.com/product/smartio/C104H.htm>
20. <http://www.rad.com/networks/1995/rs232/back.htm#backhdr>

APPENDIX A

1. FTRANSM.PAS

```
(*****
FTransM.pas
-- Main window for file transfer example program.
*****)

unit FTransM;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, ExGlobal, StdCtrls, Buttons, ExtCtrls, Printers, ComCtrls, OleCtrls,
  ToolWin, WinProcs, WinTypes, ExtDlgs, Mask, FileCtrl, Office_Tlb, Excel_TLB,
  ActnList, Grids, Outline, ChildWin;

type

  TFTransForm = class(TExampleForm)

    OpenDlg: TOpenDialog; //First Open Dialog it is used for NC Codes
    SaveDlg: TSaveDialog; //First Save Dialog used for Saving NC Codes
    StatusBar1: TStatusBar;
    Timer1: TTimer;
    PrintDialog: TPrintDialog;
    PrinterSetupDialog: TPrinterSetupDialog;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    FileNew: TMenuItem;
    FileSave: TMenuItem;
    FileSaveAs: TMenuItem;
    N2: TMenuItem;
    FilePrint: TMenuItem;
    FilePrintSetup: TMenuItem;
    N1: TMenuItem;
    FileExit: TMenuItem;
```

Port1: TMenuItem;
PortOpen: TMenuItem;
PortClose: TMenuItem;
Setting1: TMenuItem;
cmFtrans: TMenuItem;
Help1: TMenuItem;
HelpAbout: TMenuItem;
CoolBar1: TCoolBar;
ToolBar1: TToolBar;
EnterPort: TSpeedButton;
ExitPort: TSpeedButton;
Saving: TBitBtn;
Setting: TSpeedButton;
Print: TBitBtn;
SolidWorks: TBitBtn;
Words: TBitBtn;
Internet: TBitBtn;
ExitAll: TSpeedButton;
Edit1: TMenuItem;
N4: TMenuItem;
Paste1: TMenuItem;
Copy1: TMenuItem;
OLEEdit: TMenuItem;
SpeedButton6: TSpeedButton;
NCCut: TSpeedButton;
NCCopy: TSpeedButton;
NCPaste: TSpeedButton;
Comm: TBitBtn;
NCOpen: TBitBtn;
NCNew: TBitBtn;
FileOpen: TMenuItem;
Cut: TMenuItem;
ActionList1: TActionList;
Panel1: TPanel;
Outline1: TOutline;
NewBtn: TBitBtn;
PageControl1: TPageControl;
TabSheet1: TTabSheet;

```
rgProtocol: TRadioGroup;
rgDirection: TRadioGroup;
FSetOk: TBitBtn;
FSetCancel: TBitBtn;
TabSheet2: TTabSheet;
Label1: TLabel;
ListFilesLabel: TLabel;
DrivesLabel: TLabel;
DirLabel: TLabel;
FileNameLabel: TLabel;
DirBox: TDirectoryListBox;
DirDlgOK: TBitBtn;
DirDlgCancel: TBitBtn;
DriveBox: TDriveComboBox;
DirDlgPrev: TBitBtn;
FileListBox: TFileListBox;
FileNameEdit: TEdit;
FilterComboBox: TFilterComboBox;
TabSheet3: TTabSheet;
TPort: TLabel;
TFileSize: TLabel;
TProtocol: TLabel;
TLength: TLabel;
TFileName: TLabel;
Bevel1: TBevel;
Bevel2: TBevel;
Bevel3: TBevel;
Bevel4: TBevel;
Bevel5: TBevel;
lbFname: TLabel;
lbFSize: TLabel;
lbLen: TLabel;
lbPort: TLabel;
lbProtocol: TLabel;
TCancel: TBitBtn;
PropBtn: TBitBtn;
Outline2: TOutline;
OpenBtn: TBitBtn;
```

```
SaveBtn: TBitBtn;
SaveDialog2: TSaveDialog;
SaveAsSet: TMenuItem;
OpenDialog2: TOpenDialog;
ActivateOLE: TMenuItem;
N5: TMenuItem;
OLESave: TMenuItem;
SaveAsOLE: TMenuItem;
CopyOLE: TMenuItem;
PasteOLE: TMenuItem;
OpenDialog1: TOpenDialog;
SaveDialog1: TSaveDialog;
PrintDialog1: TPrintDialog;
PrinterSetupDialog1: TPrinterSetupDialog;

procedure FormCreate(Sender: TObject);
procedure SwitchMenu;
function OpenPort:Boolean;
procedure ClosePort;
function PortSet:boolean;
procedure XmitFile;
procedure RecvFile;
procedure cmFTTransClick(Sender: TObject);
procedure FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
procedure Timer1Timer(Sender: TObject);
procedure FileNewClick(Sender: TObject);
procedure FileSaveClick(Sender: TObject);
procedure FileSaveAsClick(Sender: TObject);
procedure FilePrintSetupClick(Sender: TObject);
procedure FileExitClick(Sender: TObject);
procedure PortOpenClick(Sender: TObject);
procedure PortCloseClick(Sender: TObject);
procedure Setting1Click(Sender: TObject);
procedure InternetClick(Sender: TObject);
procedure SolidWorksClick(Sender: TObject);
procedure WordsClick(Sender: TObject);
procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
```

```
Y: Integer);
procedure ControlBar1MouseMove(Sender: TObject; Shift: TShiftState; X,
Y: Integer);
procedure CommClick(Sender: TObject);
procedure FSetOkClick(Sender: TObject);
procedure FSetCancelClick(Sender: TObject);
procedure FileListBoxDbClick(Sender: TObject);
procedure TCancelClick(Sender: TObject);
procedure DirDlgOKClick(Sender: TObject);
procedure RefreshDlg(xlen:LongInt;flen:LongInt;fname:string);
procedure DirDlgCancelClick(Sender: TObject);
procedure DirDlgPrevClick(Sender: TObject);
procedure HelpAboutClick(Sender: TObject);
procedure FileOpenClick(Sender: TObject);
procedure Copy1Click(Sender: TObject);
procedure Paste1Click(Sender: TObject);
procedure CutClick(Sender: TObject);
procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
procedure NewBtnClick(Sender: TObject);
procedure Outline1MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure PropBtnClick(Sender: TObject);
procedure SaveBtnClick(Sender: TObject);
procedure SaveAsSetClick(Sender: TObject);
procedure OpenBtnClick(Sender: TObject);
procedure Outline2DragDrop(Sender, Source: TObject; X, Y: Integer);
procedure Outline2DragOver(Sender, Source: TObject; X, Y: Integer;
State: TDragState; var Accept: Boolean);
procedure Outline2DbClick(Sender: TObject);
procedure ActivateOLEClick(Sender: TObject);
procedure SaveAsOLEClick(Sender: TObject);
procedure OLESaveClick(Sender: TObject);
procedure CopyOLEClick(Sender: TObject);
procedure PasteOLEClick(Sender: TObject);
procedure FilePrintClick(Sender: TObject);
```

private

```
Falcon : String; //Used as a transitional variable in between pages
```

```
b_busy : Boolean;
DoTra : Boolean;
Olemenu: Boolean;

X1, Y1: Integer;      {mouse position}
FOLEFilename: String;
Counter: Integer;
FStartDrag: Boolean;
procedure SetOLEFileName(const Value: string);

public
V: Variant;
S: Variant;
FileName: String;

procedure ShowHint(Sender: TObject);
procedure CheckCapslock;
procedure CheckInslock;
procedure CheckNumlock;
property OLEFileName: string read FOLEFileName write SetOLEFileName;
procedure UpdateFT;
procedure UpdateGT;
procedure UpdateHt;
procedure UpdateJt;
function SaveChanges: Boolean;
function Save: Boolean;
function SaveAs: Boolean;
function SaveSetting: Boolean;
function SaveAsSetting: Boolean;
end;

var
  FTransform: TFTransform;

implementation

{$R *.DFM}
```



```
uses PComm,MxTool,Config,FtProc,About,HelpTxt,ComObj,ShellAPI,ReadThd,
SimpleM, PrintDialog, MachForm, Prop, Child;
```

```
var NumChildren: Cardinal = 0; {A var declared for Childforms}
```

```
  Lend : Integer;
```

```
  procedure TFTransForm.FormCreate(Sender: TObject);
```

```
begin
```

```
  Application.OnHint := ShowHint;
```

```
  UpdateFT; {Update File Transfer TabSheets}
```

```
{Default Settings for the Serial Port Config}
```

```
with GCommData do
```

```
begin
```

```
  Port := 3;
```

```
  ibaudrate := 14;
```

```
  iparity := 0;
```

```
  ibytesize := 3;
```

```
  istopbits := 0;
```

```
  BaudRate := B38400;
```

```
  Parity := P_NONE;
```

```
  ByteSize := BIT_8;
```

```
  StopBits := STOP_1;
```

```
  Hw := false;
```

```
  Sw := false;
```

```
  Dtr := true;
```

```
  Rts := true;
```

```
end;
```

```
{This two is used for the purpose of enabling and disabling buttons}
```

```
DoTra := false;
```

```
GszAppName := 'File Transfer Demo';
```

```
GbOpen := false;
```

```
GhForm := FTTransForm;
```

```
b_busy := false;
```

```
Olemenu := false;
```

```
SwitchMenu();
```

```
{This display the current directory as the caption of a label control}
```

```

DirBox.DirLabel := DirLabel;
{The displays the current drive so that the directory list box auomatically
updates its tree}
DriveBox.DirList := DirBox;
{This assign the edit objects name to the FileListBox's FileEdit property}
FileListBox.FileEdit := FileNameEdit;

end;

{This procedure creates a Delay function which is used in the rest of the code}
procedure Delay(ms : longint);
var
  TheTime : LongInt;
begin
  TheTime := GetTickCount + ms;
  while GetTickCount < TheTime do
    Application.ProcessMessages;
end;

procedure TFTransForm.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if ActiveMDIChild <> nil then
    CanClose := not ChildForm.Memo1.Modified or SaveChanges
end;

procedure TFTransForm.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  StatusBar1.Panels[5].Text := 'X = ' + IntToStr(X);
  StatusBar1.Panels[6].Text := 'Y = ' + IntToStr(Y);
end;

procedure TFTransForm.ControlBar1MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  StatusBar1.Panels[5].Text := 'X = ' + IntToStr(X);
  StatusBar1.Panels[6].Text := 'Y = ' + IntToStr(Y);

```

```
end;
```

```
procedure TFTransform.SwitchMenu;
```

```
begin
```

```
  EnterPort.Enabled := not GbOpen;
```

```
  PortOpen.Enabled := not GbOpen;
```

```
  PortClose.Enabled := GbOpen;
```

```
  ExitPort.Enabled := GbOpen;
```

```
  cmFTTrans.Enabled := GbOpen and (not b_busy);
```

```
end;
```

```
function TFTransform.PortSet:boolean;
```

```
var
```

```
  port : LongInt;
```

```
  mode : LongInt;
```

```
  hw,sw : LongInt;
```

```
  ret : LongInt;
```

```
begin
```

```
  port := GCommData.Port;
```

```
  mode := GCommData.Parity or GCommData.ByteSize or GCommData.StopBits;
```

```
  PortSet := false;
```

```
  if GCommData.Hw then
```

```
    hw := 3    { bit0 and bit1 }
```

```
  else
```

```
    hw := 0;
```

```
  if GCommData.Sw then
```

```
    sw := 12   { bit2 and bit3 }
```

```
  else
```

```
    sw := 0;
```

```
  ret := sio_ioctl(port,GCommData.BaudRate,mode);
```

```
  if ret<>SIO_OK then
```

```
    begin
```

```
      MxShowError('sio_ioctl',ret);
```

```
      Exit;
```

```
    end;
```

```
ret := sio_flowctrl(port,hw or sw);
if ret<>SIO_OK then
begin
  MxShowError('sio_flowctrl',ret);
  Exit;
end;

ret := sio_DTR(port,Integer(GCommData.Dtr));
if ret<>SIO_OK then
begin
  MxShowError('sio_DTR',ret);
  Exit;
end;

if not GCommData.Hw then
begin
  ret := sio_RTS(port,Integer(GCommData.Rts));
  if ret<>SIO_OK then
  begin
    MxShowError('sio_RTS',ret);
    Exit;
  end;
end;

ShowStatus();
PortSet := True;
end;

function TFTransform.OpenPort:Boolean;
var
  ret:Integer;
begin
  OpenPort := false;
  ret := sio_open(GCommData.Port);
  if ret <> SIO_OK then
  begin
    MxShowError('sio_open',ret);
```

```
Exit;
end;

if PortSet() = false then
begin
  sio_close(GCommData.Port);
  Exit;
end;
OpenPort := true;
GbOpen := true;
Outline1.Enabled := DoTra;
SwitchMenu();
ShowStatus();
end;

procedure TFTransForm.ClosePort;
begin
  sio_close (GCommData.Port);
  GbOpen := False;
  SwitchMenu();
  ShowStatus();
end;

procedure TFTransForm.Setting1Click(Sender: TObject);
var
  bakdata : TCOMMDATA;
begin
  StatusBar1.Panels[4].Text := 'Port Setting';
  bakdata := GCommData;
  if CfgForm.ShowModal = mrCancel then
    StatusBar1.Panels[4].Text := "";
  Exit;

  if GbOpen then
    if PortSet()=false then
      begin
        GCommData := bakdata;
        Exit;
```

```
end;
ShowStatus();
end;

procedure TFTransForm.PortOpenClick(Sender: TObject);
begin
  OpenPort();
  SysUtils.Beep;
  Windows.Beep( 00, 000);
  StatusBar1.Panels[4].Text := 'Open Port';
  Delay (2000);
  StatusBar1.Panels[4].Text := "";
end;

procedure TFTransForm.PortCloseClick(Sender: TObject);
begin
  ClosePort();
  SysUtils.Beep;
  Windows.Beep( 00, 000);
  StatusBar1.Panels[4].Text := 'Close Port';
  Delay (2000);
  StatusBar1.Panels[4].Text := "";
end;

procedure TFTransForm.FileNewClick(Sender: TObject);
begin
  if not ChildForm.Modified or SaveChanges then
  begin
    Inc (Counter);
    Inc(NumChildren);
    with TChildForm.Create(Application) do
      begin
        Caption := 'Untitled' + IntToStr(NumChildren);
        {Bring up insert OLE object dialog and insert into child }
        ChildForm.Memo1.Text := "";
        Modified := False;
        FileName := "";
        Caption := 'File - [Untitled]';
```

```
        end;
    end;
end;

procedure TFTransForm.FileOpenClick(Sender: TObject);
begin
    if OpenDlg.Execute then
        with TChildForm.Create(Application) do
            begin
                try
                    OleFileName := OpenDlg.FileName;
                    Memo1.Lines.LoadFromFile(OleFileName);
                    Show;
                except
                    Release; // free form on error
                    raise; // re-raise exception
                end;
            end;
        end;
end;

function TFTransForm.SaveChanges: Boolean;
begin
    case MessageDlg (
        'The document ' + filename + ' has changed.' +
        #13#13 + 'Do you want to save the changes?',
        mtConfirmation, mbYesNoCancel, 0) of
        idYes:
            // call Save and return its result
            Result := Save;
        idNo:
            // do not save and continue
            Result := True;
        else // idCancel:
            // do not save and abort operation
            Result := False;
    end;
end;
```



```

{a return value "False" means the SaveAs
operation has been aborted}
function TFTransForm.Save: Boolean;
begin
  if Filename = "" then
    Result := SaveAs // ask for a file name
  else
    begin
      ChildForm.Memo1.Lines.SaveToFile (FileName);
      ChildForm.Modified := False;
      Result := True;
    end;
end;

{return a value "False" if the SaveAs
dialog box has been 'cancelled'}
function TFTransForm.SaveAs: Boolean;
begin
  SaveDlg.FileName := Filename;
  if SaveDlg.Execute then
    begin
      Filename := SaveDlg.FileName;
      Save;
      Caption := 'RichNote - ' + Filename;
      Result := True;
    end
  else
    Result := False;
end;

procedure TFTransForm.FileSaveClick(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    { if no name is assigned, then do a "save as" }
    if TChildForm(ActiveMDIChild).OLEFileName = "" then
      FileSaveAsClick(Sender)
    else
      { otherwise save under current name }

```

```

    with TChildForm(ActiveMDIChild) do
        Memo1.Lines.SaveToFile(OLEFileName);
end;

procedure TFTransForm.FileSaveAsClick(Sender: TObject);
begin
    if (ActiveMDIChild <> nil) and (SaveDlg.Execute) then
        with TChildForm(ActiveMDIChild) do
            begin
                OleFileName := SaveDlg.FileName;
                Memo1.Lines.SaveToFile(OleFileName);
            end;
end;

procedure TFTransForm.FilePrintClick(Sender: TObject);
var
    PrintFile: TextFile;
    I: Integer;
    bakdata : TCOMMDATA;
begin
    StatusBar1.Panels[4].Text := 'Print File';
    {PrintButton.ShowModal;}
    begin
        bakdata := GCommData;
        if PrintDialog.Execute then
            begin
                AssignPrn (PrintFile);
                Rewrite (PrintFile);
                try
                    Printer.Canvas.Font:= ChildForm.Memo1.Font;
                    for I:=0 to ChildForm.Memo1.Lines.Count-1 do
                        WriteLn(PrintFile, ChildForm.Memo1.Lines[I]);
                    finally
                        CloseFile (PrintFile);
                    end;
            end;
        Delay (2000);
        StatusBar1.Panels[4].Text := "";

```

```
end;
end;

procedure TFTransForm.FilePrintSetupClick(Sender: TObject);
begin
  PrinterSetupDialog.Execute;
end;

procedure TFTransForm.FileExitClick(Sender: TObject);
begin
  Close;
end;

procedure TFTransForm.cmFTTransClick(Sender: TObject);
var
  W: Word;
begin
  PageControl1.ActivePage := TabSheet1;
  W:=MessageDlg('Activate File Transfer Protocols?', mtConfirmation,[mbYes, mbNo], 0);
  case W of
    mrYes : UpdateGT;
    mrNo : Exit;
  end;
end;

procedure TFTransForm.CutClick(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    TChildForm(ActiveMDIChild).Memo1.CutToClipboard;
end;

procedure TFTransForm.Copy1Click(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    TChildForm(ActiveMDIChild).Memo1.CopyToClipboard;
end;

procedure TFTransForm.Paste1Click(Sender: TObject);
```

```
begin
  if ActiveMDIChild <> nil then
    TChildForm(ActiveMDIChild).Memo1.PasteFromClipboard;
  end;

procedure TFTransForm.SetOLEFileName(const Value: string);
begin
  if Value <> FOLEFileName then
    begin
      FOLEFileName := Value;
      Caption := ExtractFileName(FOLEFileName);
    end;
end;

procedure TFTransForm.ActivateOLEClick(Sender: TObject);
begin
  if not ChildForm.Modified or SaveChanges then
    begin
      Inc(Counter);
      Inc(NumChildren);
      with TMDIChild.Create(Application) do
        begin
          Caption := 'Untitled' + IntToStr(NumChildren);
          { bring up insert OLE object dialog and insert into child }
          OleContainer.InsertObjectDialog;
        end;
      end;
    end;
end;

procedure TFTransForm.SaveAsOLEClick(Sender: TObject);
begin
  if (ActiveMDIChild <> nil) and (SaveDialog1.Execute) then
    with TMDIChild(ActiveMDIChild) do
      begin
        OleFileName := SaveDialog1.FileName;
        OleContainer.SaveToFile(OleFileName);
      end;
    end;
end;
```

```

procedure TFTransForm.OLESaveClick(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    { if no name is assigned, then do a "save as" }
    if TMDIChild(ActiveMDIChild).OLEFileName = "" then
      SaveAsOLEClick(Sender)
    else
      { otherwise save under current name }
      with TMDIChild(ActiveMDIChild) do
        OleContainer.SaveToFile(OLEFileName);
end;

```

```

procedure TFTransForm.CopyOLEClick(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    TMDIChild(ActiveMDIChild).OleContainer.Copy;
end;

```

```

procedure TFTransForm.PasteOLEClick(Sender: TObject);
begin
  if ActiveMDIChild <> nil then
    with TMDIChild(ActiveMDIChild).OleContainer do
      { Before invoking dialog, check to be sure that there }
      { are valid OLE objects on the clipboard. }
      if CanPaste then PasteSpecialDialog;
end;

```

```

procedure TFTransForm.HelpAboutClick(Sender: TObject);
begin
  AboutFrm.ShowModal;
end;

```

```

procedure TFTransForm.CommClick(Sender: TObject);
var
  W: Word;
begin
  if EnterPort.Enabled = not DoTra then

```

```
begin
W:=MessageDlg('Port is not Opened? Continue Accessing Communication Dialog?',
mtConfirmation,[mbYes, mbNo], 0);
  case W of
    mrYes :
      begin
        PortOpen.Click;
        GhExit := false;
        TReadThread.Create(false);
        SimpleForm.ShowModal;
      end;
    mrNo ;;
  end;
end
else
GhExit := false;
TReadThread.Create(false);
SimpleForm.ShowModal;
end;
```

```
procedure TFTransForm.SolidWorksClick(Sender: TObject);
begin
  StatusBar1.Panels[4].Text := 'Loading SolidWorks';
  V := CreateOleObject('SldWorks.Application');
  V.Visible :=True;
  Delay (2000);
  StatusBar1.Panels[4].Text := "";
end;
```

```
procedure TFTransForm.WordsClick(Sender: TObject);
begin
  StatusBar1.Panels[4].Text := 'Load Microsoft Word';
  S := CreateOleObject('Word.Application');
  S.Visible :=True;
  Delay (2000);
  StatusBar1.Panels[4].Text := "";
end;
```

```
procedure TFTransForm.InternetClick(Sender: TObject);
Var St:Array[0..255] of char;
begin
  StatusBar1.Panels[4].Text := 'Access Internet';
  ShellExecute(Handle,'open',StrPCopy(St,'http://'+Internet.Caption),nil,nil,SW_SHOW);
  Delay (2000);
  StatusBar1.Panels[4].Text := "";
end;

procedure TFTransForm.ShowHint(Sender: TObject);
begin
  StatusBar1.Panels[0].Text := Application.Hint;
end;

procedure TFTransForm.CheckCapslock;
begin
  if Odd (GetKeyState (VK_CAPITAL)) then
    StatusBar1.Panels[1].Text := 'CAPS'
  else
    StatusBar1.Panels[1].Text := "";
end;

procedure TFTransForm.CheckInslock;
begin
  if Odd (GetKeyState (VK_INSERT)) then
    StatusBar1.Panels[2].Text :='INS'
  else
    StatusBar1.Panels[2].Text :="";
end;

procedure TFTransForm.CheckNumlock;
begin
  if Odd (GetKeyState (VK_NUMLOCK)) then
    StatusBar1.Panels[3].Text :='NUM'
  else
    StatusBar1.Panels[3].Text :="";
end;
```



```
procedure TFTransForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  CheckCapslock;
  CheckInslock;
  CheckNumlock;
end;
```

```
procedure TFTransForm.Timer1Timer(Sender: TObject);
begin
  CheckCapslock;
  CheckInslock;
  CheckNumlock;
  StatusBar1.Panels[8].Text := TimeToStr(Time);
end;
```

```
procedure TFTransForm.NewBtnClick(Sender: TObject);
begin
  MachineForm.ShowModal;
end;
```

```
procedure TFTransForm.Outline1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  fStartDrag: Boolean;
begin
  fStartDrag := True;
  if (Outline1.ItemCount > 0) and (Button = mbLeft) then
    Outline1.BeginDrag (True);
  fStartDrag := False;
end;
```

```
procedure TFTransForm.PropBtnClick(Sender: TObject);
begin
  PropertyForm.ShowModal;
end;
```

```
function TFTransForm.SaveSetting: Boolean;
```

```

begin
  if Filename = "" then
    Result := SaveAsSetting // ask for a file name
  else
    begin
      Outline1.Lines.SaveToFile (FileName);
      ChildForm.Modified := False;
      Result := True;
    end;
end;

function TFTransForm.SaveAsSetting: Boolean;
begin
  SaveDialog2.FileName := Filename;
  if SaveDialog2.Execute then
    begin
      Filename := SaveDialog2.FileName;
      SaveSetting;
      Caption := 'RichNote - ' + Filename;
      Result := True;
    end
  else
    Result := False;
end;

procedure TFTransForm.SaveBtnClick(Sender: TObject);
begin
  StatusBar1.Panels[4].Text := 'Save File';
  begin
    if Filename = "" then
      SaveAsSetClick(Sender) // ask for a file name
    else
      begin
        if ChildForm.Modified then
          SaveSetting;
        end;
      end;
  end;
  Delay (2000);
end;

```

```
StatusBar1.Panels[4].Text := "";
end;

procedure TFTransForm.SaveAsSetClick(Sender: TObject);
begin
    SaveAsSetting;
end;

procedure TFTransForm.OpenBtnClick(Sender: TObject);
begin
    if not ChildForm.Modified or SaveChanges then
    if OpenFileDialog2.Execute then
    begin
        FileName := OpenFileDialog2.FileName;
        Outline1.Lines.LoadFromFile (FileName);
        ChildForm.Modified := false;
        Caption := FileName;
    end;
end;
end;

procedure TFTransForm.Outline2DragDrop(Sender, Source: TObject; X,
Y: Integer);
var
    Current: Integer;
begin
    Current := Outline2.GetItem (X, Y);
    if Current > 0 then
    begin
        Outline2.AddChild (Current, Outline1.Lines[Outline1.SelectedItem - 1]);
        Outline2.Items [Current].Expanded := True;
    end
    else
        MessageDlg ('You've not dragged over an item',
            mtError, [mbOk], 0);
end;
```

```

procedure TFTransForm.Outline2DragOver(Sender, Source: TObject; X,
  Y: Integer; State: TDragState; var Accept: Boolean);
begin
  if Sender is TOutline then
    Accept := True;
end;

```

```

procedure TFTransForm.Outline2DbClick(Sender: TObject);
begin
  if CfgForm.ShowModal = mrYes then
    OpenPort();
    UpdateGT;
end;

```

```

procedure TFTransForm.FSetOkClick(Sender: TObject);
begin
  //Declare which protocol to use.
  GProtocol := FTTransForm.rgProtocol.ItemIndex;
  //Declare whether to send or to receive file.
  GDirection := FTTransForm.rgDirection.ItemIndex;
  //Update of the buttons and sheets.
  UpdateHt;

  if FTTransForm.rgDirection.ItemIndex = FT_XMIT then
    //Access the the second page of Tab.
    PageControl1.ActivePage := TabSheet2
  else
    PageControl1.ActivePage := TabSheet2;
end;

```

```

procedure TFTransForm.FSetCancelClick(Sender: TObject);
begin
  UpdateFt;
end;

```

```

procedure TFTransForm.FileListBoxDbClick(Sender: TObject);
begin
  Falcon := FileNameEdit.Text;

```

```

if FTTransForm.rgDirection.ItemIndex = FT_XMIT then
  XmitFile
else
  begin
    if (GProtocol=FTZMDM) or (GProtocol=FTYMDM) or (GProtocol=FTKERMIT)then
      begin
        Istrcpy(GrPath,PChar(FTTransForm.DirBox.Directory));
        SetCurrentDir(GrPath)
      end
    else
      begin
        Istrcpy(GrFname,PChar(Falcon));
      end;
      PageControl1.ActivePage :=TabSheet3;
      RecvFile;
    end;
  end;

procedure TFTTransForm.DirDlgOKClick(Sender: TObject);
begin
  Falcon := FileNameEdit.Text;
  if FTTransForm.rgDirection.ItemIndex = FT_XMIT then
    XmitFile
  else
    begin
      if (GProtocol=FTZMDM) or (GProtocol=FTYMDM) or (GProtocol=FTKERMIT)then
        begin
          Istrcpy(GrPath,PChar(FTTransForm.DirBox.Directory));
          SetCurrentDir(GrPath)
        end
      else
        begin
          Istrcpy(GrFname,PChar(Falcon));
        end;
      PageControl1.ActivePage :=TabSheet3;
      RecvFile;
    end;
  end;
end;

```

```

procedure TFTransForm.DirDlgCancelClick(Sender: TObject);
begin
    GftCancel :=true;
    PageControl1.ActivePage := TabSheet1;
    UpdateFt;
end;

```

```

procedure TFTransForm.DirDlgPrevClick(Sender: TObject);
begin
    {if Previous button is pushed the program returns to TabSheet 1}
    PageControl1.ActivePage := TabSheet1;
end;

```

```

procedure TFTransForm.TCancelClick(Sender: TObject);
begin
    GftCancel :=true;
    PageControl1.ActivePage := TabSheet1;
    UpdateFt;
end;

```

```

procedure TFTransForm.RefreshDlg(xlen:LongInt;flen:LongInt;fname:string);
begin
    lbFSize.Caption := IntToStr(flen);
    lbPort.Caption := IntToStr(GCommData.Port);
    lbProtocol.Caption := GstrProtocol[GProtocol];
    lbFName.Caption := fname;
    lbxLen.Caption := IntToStr(xlen);
end;

```

```

procedure TFTransForm.XmitFile;
begin
    {Declare Falcon using the FileNameEdit string}
    Falcon := FileNameEdit.Text;
    {Istrcpy copies the entire contents of one string into another string.
    Either string, instead of being a "real" string, can also be merely a pointer to a string instead.
    The target string must already have enough space to receive the source string's contents.
    The function also will copy a terminating null character into the target string}

```

```

Istrcpy(GxFname,PChar(Falcon));
{If user press 'Cancel' button which on status dialog,
'GftCancel' flag will be set to true.This will let callback
function to return -1 to terminate file transfer.}
GftCancel := false;
TFtProc.Create(false);
PageControl1.ActivePage := TabSheet3;
UpdateJt;
end;

```

```

procedure TFTransForm.RecvFile;
begin
    GftCancel := false;
    TFtProc.Create(false);
end;

```

{The first menu button updates, most button are disabled until transfer config are decalred}

```

procedure TFtransForm.UpdateFT;
begin
    PageControl1.ActivePage:=TabSheet1;
    NCPaste.Enabled := DoTra;
    NCCut.Enabled := DoTra;
    NCCopy.Enabled := DoTra;
    FSetOk.Enabled := DoTra;
    FSetCancel.Enabled := DoTra;
    rgProtocol.Enabled := DoTra;
    rgDirection.Enabled := DoTra;
    TabSheet1.Enabled := DoTra;
    TabSheet2.Enabled := DoTra;
    TabSheet3.Enabled := DoTra;
    DirDlgOK.Enabled := DoTra;
    DirDlgCancel.Enabled := DoTra;
    DirDlgPrev.Enabled := DoTra;
    TCancel.Enabled := DoTra;
    lbFSize.Caption := "";
    lbPort.Caption := "";
    lbProtocol.Caption := "";
    lbFName.Caption := "";

```



```
    lbxLen.Caption := " ";  
end;
```

```
{This is the update after Config is declared, only TabSheet1 is allowed}
```

```
procedure TFtransForm.UpdateGT;  
begin  
    FSetOk.Enabled := not DoTra;  
    FSetCancel.Enabled := not DoTra;  
    TabSheet1.Enabled := not DoTra;  
    rgProtocol.Enabled := not DoTra;  
    rgDirection.Enabled := not DoTra;  
    TabSheet2.Enabled := DoTra;  
    TabSheet3.Enabled := DoTra;  
    DirDlgOK.Enabled := DoTra;  
    DirDlgCancel.Enabled := DoTra;  
    DirDlgPrev.Enabled := DoTra;  
    TCancel.Enabled := DoTra;  
end;
```

```
{The second update, after the direction and transfer protocol is declared}
```

```
procedure TFtransForm.UpdateHt;  
begin  
    TabSheet2.Enabled := not DoTra;  
    TabSheet3.Enabled := DoTra;  
    DirDlgOK.Enabled := not DoTra;  
    DirDlgCancel.Enabled := not DoTra;  
    DirDlgPrev.Enabled := not DoTra;  
    TabSheet3.Enabled := DoTra;  
    TCancel.Enabled := DoTra;  
end;
```

```
{The Thrid update, after the file storage location have been specified}
```

```
procedure TFtransForm.UpdateJt;  
begin  
    TabSheet3.Enabled := not DoTra;  
    FSetOk.Enabled := DoTra;  
    DirDlgOK.Enabled := DoTra;  
    DirDlgPrev.Enabled := DoTra;
```

```
FSetOk.Enabled := DoTra;  
FSetCancel.Enabled := DoTra;  
rgProtocol.Enabled := DoTra;  
rgDirection.Enabled := DoTra;  
TCancel.Enabled := not DoTra;  
end;  
  
end.
```

2. FOUNTERA_TLB.PAS

```

unit FounTera_TLB;

// ***** //
// WARNING //
// ----- //
// The types declared in this file were generated from data read from a //
// Type Library. If this type library is explicitly or indirectly (via //
// another type library referring to this type library) re-imported, or the //
// 'Refresh' command of the Type Library Editor activated while editing the //
// Type Library, the contents of this file will be regenerated and all //
// manual modifications will be lost. //
// ***** //

// PASTLWTR : $Revision: 1.11.1.63 $
// File generated on 1/11/2001 7:09:02 p.m. from Type Library described below.

// ***** //
// Type Lib: C:\Documents and Settings\Captainshaft\My
Documents\Masters\FounTera\FounTera.tlb
// IID\LCID: {F8F19781-DB75-4ED4-85C3-F5B0EE44C5AA}\0
// Helpfile:
// HelpString: FounTera Library
// Version: 1.0
// ***** //

interface

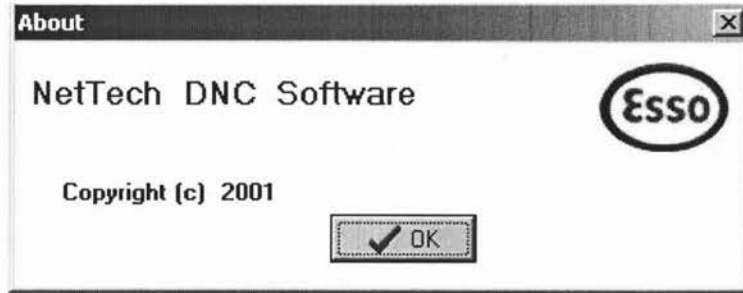
uses Windows, ActiveX, Classes, Graphics, OleCtrls, StdVCL;

// *****//
// GUIDS declared in the TypeLibrary. Following prefixes are used: //
// Type Libraries : LIBID_xxxx //
// CoClasses : CLASS_xxxx //
// DISPInterfaces : DIID_xxxx //
// Non-DISP interfaces: IID_xxxx //

```

```
// *****//  
const  
  LIBID_FounTera: TGUID = '{F8F19781-DB75-4ED4-85C3-F5B0EE44C5AA}';  
  
implementation  
  
uses ComObj;  
  
end.
```

3. ABOUT.PAS



```
(*****
```

```
About.pas
```

```
*****)
```

```
unit About;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Buttons, ExtCtrls, jpeg;
```

```
type
```

```
TAboutFrm = class(TForm)
```

```
  AboutLabel: TLabel;
```

```
  Label2: TLabel;
```

```
  BitBtn1: TBitBtn;
```

```
  Image1: TImage;
```

```
  procedure AboutOKClick(Sender: TObject);
```

```
  procedure BitBtn1Enter(Sender: TObject);
```

```
private
```

```
  { Private declarations }
```

```
public
```

```
  { Public declarations }
```

```
end;
```

```
var
  AboutFrm: TAboutFrm;

implementation

uses ExGlobal;
{$R *.DFM}

procedure TAboutFrm.AboutOKClick(Sender: TObject);
begin
  Close();
end;

procedure TAboutFrm.BitBtn1Enter(Sender: TObject);
begin
  AboutLabel.Caption := 'NetTech '+' DNC '+' Software ';
end;

end.
```

4. CHILD.PAS



```
unit Child;
```

```
interface
```

```
uses SysUtils, Windows, Messages, Classes, Graphics, Controls,
    Forms, Dialogs, OleCtrls, StdCtrls;
```

```
type
```

```
TChildForm = class(TForm)
```

```
    Memo1: TMemo;
```

```
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
```

```
    procedure Memo1MouseMove(Sender: TObject; Shift: TShiftState; X,
        Y: Integer);
```

```
    procedure Memo1Click(Sender: TObject);
```

```
    procedure Memo1Enter(Sender: TObject);
```

```
    procedure Memo1Exit(Sender: TObject);
```

```
private
```

```
    FOLEFilename: string;
```

```
    DoTra : Boolean;
```

```
    Lend : Integer;
```

```
    procedure SetOLEFileName(const Value: string);
```

```
public
```

```
    Modified : Boolean;
```

```
    property OLEFileName: string read FOLEFilename write SetOLEFileName;
```

```
end;
```

```

var
  ChildForm: TChildForm;

implementation

uses FTransM;

{$R *.DFM}
procedure Delay(ms : longint);
var
  TheTime : LongInt;
begin
  TheTime := GetTickCount + ms;
  while GetTickCount < TheTime do
    Application.ProcessMessages;
end;

procedure TChildForm.SetOLEFileName(const Value: string);
begin
  if Value <> FOLEFileName then

  begin
    FOLEFileName := Value;
    Caption := ExtractFileName(FOLEFileName);
    //ExtractFileNam
    e
  end;
end;

procedure TChildForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if Memo1.Modified and (Length (Memo1.lines.Text)>0) then
    FtransForm.FileSaveAsClick (sender);
  Action := Cafree;
  FtransForm.NCPaste.Enabled := DoTra;
  FtransForm.NCCut.Enabled := DoTra;

```



```
FtransForm.NCCopy.Enabled := DoTra;
end;

procedure TChildForm.Memo1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  FtransForm.StatusBar1.Panels[5].Text := 'X = ' + IntToStr(X);
  FtransForm.StatusBar1.Panels[6].Text := 'Y = ' + IntToStr(Y);
end;

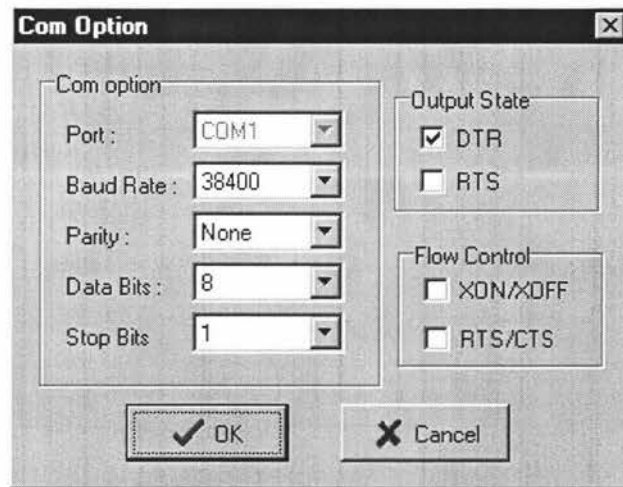
procedure TChildForm.Memo1Click(Sender: TObject);
begin
  FtransForm.StatusBar1.Panels[4].Text := 'NC Editor';
  if Memo1.SelLength > 0 then
    FtransForm.NCCut.Enabled := not DoTra;
  if Memo1.SelLength > 0 then
    FtransForm.NCCopy.Enabled := not DoTra;
  if Memo1.SelLength = 0 then
    FtransForm.NCCut.Enabled := DoTra;
  if Memo1.SelLength = 0 then
    FtransForm.NCCopy.Enabled := DoTra;
  Delay (2000);
  FtransForm.StatusBar1.Panels[4].Text := '';
end;

procedure TChildForm.Memo1Enter(Sender: TObject);
begin
  FtransForm.NCPaste.Enabled := not DoTra;
end;

procedure TChildForm.Memo1Exit(Sender: TObject);
begin
  FtransForm.NCPaste.Enabled := DoTra;
end;

end.
```

5. CONFIG.PAS



(*****

Config.pas

-- Config dialog for com port commnucation parameters

*****)

unit Config;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Buttons;

type

TCfgForm = class(TForm)
 GroupBox1: TGroupBox;
 Label1: TLabel;
 cbPort: TComboBox;
 Label2: TLabel;
 cbBaudRate: TComboBox;
 Label3: TLabel;
 cbParity: TComboBox;

```
Label4: TLabel;  
cbByteSize: TComboBox;  
Label5: TLabel;  
cbStopBits: TComboBox;  
GroupBox2: TGroupBox;  
chHw: TCheckBox;  
chSw: TCheckBox;  
GroupBox3: TGroupBox;  
chDtr: TCheckBox;  
chRts: TCheckBox;  
Cancel: TBitBtn;  
OK: TBitBtn;  
  
procedure FormCreate(Sender: TObject);  
procedure chHwClick(Sender: TObject);  
procedure CfgCancelClick(Sender: TObject);  
procedure FormActivate(Sender: TObject);  
procedure OKClick(Sender: TObject);  
  
private  
  { Private declarations }  
public  
  { Public declarations }  
end;  
  
var  
  CfgForm: TCfgForm;  
  
implementation  
  
uses ExGlobal;  
  
var  
  Gfhw : boolean;  
  
{ $R *.DFM }  
  
procedure TCfgForm.FormCreate(Sender: TObject);
```

```
var
  i:Word;
begin
  for i:=1 to 256 do
    cbPort.Items.Add('COM'+IntToStr(i));
  end;

procedure TCfgForm.FormActivate(Sender: TObject);
begin
  with GCommData do
  begin
    cbPort.ItemIndex := Port-1;

    cbBaudRate.ItemIndex := ibaudrate;
    cbParity.ItemIndex := iparity;
    cbByteSize.ItemIndex := ibratesize;
    cbStopBits.ItemIndex := istopbits;

    chHw.Checked := Hw;
    chSw.Checked := Sw;
    chRts.Checked := Rts;
    chDtr.Checked := Dtr;

    Gfhw := Hw;
    chRts.Enabled := not Gfhw;

    { disable com port setting when open }
    cbPort.Enabled := not GbOpen;
  end;
end;

procedure TCfgForm.chHwClick(Sender: TObject);
begin
  chRts.Enabled := Gfhw;
  Gfhw := Not Gfhw;
end;

procedure TCfgForm.CfgCancelClick(Sender: TObject);
```

```
begin
  ModalResult := mrCancel;
end;

procedure TCfgForm.OKClick(Sender: TObject);
var W:Word;
begin
  with GCommData do
    begin
      Port := cbPort.ItemIndex + 1;

      ibaudrate := cbBaudRate.ItemIndex;
      iparity := cbParity.ItemIndex;
      ibratesize := cbByteSize.ItemIndex;
      istopbits := cbStopBits.ItemIndex;

      BaudRate := GBaudTable[ibaudrate];
      ByteSize := GByteSizeTable[ibratesize];
      Parity := GParityTable[iparity];
      StopBits := GStopBitsTable[istopbits];

      Hw := chHw.Checked;
      Sw := chSw.Checked;
      Rts := chRts.Checked;
      Dtr := chDtr.Checked;
    end;

    begin
      W:=MessageDlg('Settings Correct?', mtConfirmation,[mbYes, mbNo], 0);
      case W of
        mrYes: ModalResult := mrOk;
        mrNo: ModalResult := mrCancel;
      end;
    end;
  end;
end.
```

6. EXGLOBAL.PAS

(*****

ExGlobal.pas

-- Global variable & Com port record defined for
example program.

*****)

unit ExGlobal;

interface

uses Forms,Menus,StdCtrls,PComm;

type

TExampleForm = class(TForm)

 Term: TMemo;

 procedure ShowStatus;

private

 { Private declarations }

protected

public

 { Public declarations }

end;

TCOMMDATA = record

 Port : LongInt;

 BaudRate : Integer;

 Parity : Integer;

 ByteSize : Integer;

 StopBits : Integer;

 ibaudrate : Integer;

 iparity : Integer;

 ibytesize : Integer;

 istopbits : Integer;

 Hw : boolean;

 Sw : boolean;

```

Dtr : boolean;
Rts : boolean;
hNC : boolean;
end;

var
{ Global variable for example }
GCommData : TCOMMDATA;
GszAppName : string;
GhForm   : TExampleForm; { main form }
GbOpen   : boolean;      { opened ? }
GhExit   : boolean;      { stop thread ? }

GBaudTable :array[0 .. 19] of Integer = (
    B50,B75,B110,B134,B150,B300,B600,B1200,
    B1800,B2400,B4800,B7200,B9600,B19200,B38400,
    B57600,B115200,B230400,B460800,B921600
);

GParityTable :array[0..4] of Integer = (
    P_NONE,P_EVEN,P_ODD,P_MRK,P_SPC
);
GByteSizeTable:array[0..3] of Integer =(
    BIT_5,BIT_6,BIT_7,BIT_8
);

GStopBitsTable:array[0..1] of Integer = (
    STOP_1,STOP_2
);

GstrBaudTable :array[0..19] of string = (
    '50','75','110','134','150','300',
    '600','1200','1800','2400','4800','7200',
    '9600','19200','38400','57600','115200',
    '230400','460800','921600'
);

GstrParityTable :array[0..4] of string = (

```

```
'None','Odd','Even','Mark','Space'
);

GstrByteSizeTable:array[0..3] of string = (
  '5','6','7','8'
);

GstrStopBitsTable:array[0..1] of string = (
  '1','2'
);
```

implementation

uses SysUtils;

```
procedure TExampleForm.ShowStatus;
var
  szMessage : string;
begin
  szMessage := GszAppName;
  if GbOpen then
  begin
    with GCommData do
    begin
      szMessage := szMessage + ' -- COM' + IntToStr(Port) + ',';
      szMessage := szMessage +
        GstrBaudTable[ibaudrate] + ',';
      szMessage := szMessage +
        GstrParityTable[iparity] + ',';
      szMessage := szMessage +
        GstrByteSizeTable[ibyteSize] + ',';
      szMessage := szMessage +
        GstrStopBitsTable[istopbits];
      if Hw then
        szMessage := szMessage + ',RTS/CTS';
      if Sw then
        szMessage := szMessage + ',XON/XOFF';
    end;
  end;
```



```
end;  
Caption := szMessage;  
end;  
  
end.
```

7. FTPROC.PAS

```
(*****
  FtProc.pas
  -- File transfer thread for file transfer example program.
  *****)
```

```
unit FtProc;
```

```
interface
```

```
uses
```

```
  Classes;
```

```
Const
```

```
  FT_XMIT = 0;
```

```
  FT_RECV = 1;
```

```
  FTXMDM1KCRC = 0;
```

```
  FTXMDMCHK = 1;
```

```
  FTXMDMCRC = 2;
```

```
  FTZMDM = 3;
```

```
  FTYMDM = 4;
```

```
  FTKERMIT = 5;
```

```
  FTASCII = 6;
```

```
  MAX_PATH = 260; {Win32 defined}
```

```
type
```

```
  TFtProc = class(TThread)
```

```
  private
```

```
    { Private declarations }
```

```
  protected
```

```
    procedure Execute; override;
```

```
  end;
```

```
var
```

```
  GxFname : array[0..MAX_PATH] of Char;
```

```
  GrFname : array[0..MAX_PATH] of Char;
```

```
GrPath : array[0..MAX_PATH] of Char;
```

```
GstrProtocol:array [0..6] of string = (
  'XModem-1KCRC','XModem-CheckSum','XModem-CRC',
  'ZModem','YModem','Kermit','ASCII'
);
```

```
GProtocol : Word;
```

```
GDirection : Word;
```

```
GftCancel : boolean;
```

implementation

uses Windows,Forms,PCComm,ExGlobal,FTTransM,MxTool, Comobj;

```
function xCallBack(xmitlen:LongInt;bufen:LongInt;buf:PChar;flen:LongInt):
  LongInt;stdcall;forward;
```

```
function rCallBack(recvlen:LongInt;bufen:LongInt;buf:PChar;flen:LongInt):
  LongInt;stdcall;forward;
```

```
procedure ProcessRet(port:LongInt;ret:LongInt;protocol:Word;direction:Word);
  forward;
```

```
{ TFtProc }
```

```
(*
```

After create thread object in main process,'Execute()' function will be called automatically.

If user press 'Cancel' button which on status dialog, 'GftCancel' flag will be set to true.This will let callback function to return -1 to terminate file transfer.

```
*)
```

```
procedure TFtProc.Execute;
```

```
var
```

```
  ret : LongInt;
```

```
  port: LongInt;
```

```
  fname : PChar;
```

```
begin
```

```
  { Place thread code here }
```

```

port := GCommData.Port;

ret := 0;
if (GDirection = FT_XMIT) then
begin
  case GProtocol of
    FTXMDM1KCRC:
      ret := sio_FtXmodem1KCRCTx(port,GxFname,xCallBack, 27);
    FTXMDMCHK:
      ret := sio_FtXmodemCheckSumTx(port,GxFname,xCallBack, 27);
    FTXMDMCRC:
      ret := sio_FtXmodem1KCRCTx(port,GxFname,xCallBack, 27);
    FTZMDM:
      ret := sio_FtZmodemTx(port,GxFname,xCallBack, 27);
    FTYMDM:
      ret := sio_FtYmodemTx(port,GxFname,xCallBack, 27);
    FTKERMIT:
      ret := sio_FtKermitTx(port,GxFname,xCallBack, 27);
    FTASCII:
      ret := sio_FtASCIITx(port,GxFname,xCallBack, 27);
  end;
end
else {FT_RECV}
begin
  case GProtocol of
    FTXMDM1KCRC:
      ret := sio_FtXmodem1KCRCRx(Port, GrFname,rCallBack, 27);
    FTXMDMCHK:
      ret := sio_FtXmodemCheckSumRx(Port, GrFname,rCallBack, 27);
    FTXMDMCRC:
      ret := sio_FtXmodem1KCRCRx(Port, GrFname,rCallBack, 27);
    FTZMDM:
      begin
        fname := GrFname;
        ret := sio_FtZmodemRx(Port, fname,1,rCallBack, 27);
      end;
    FTYMDM:
      begin

```

```

    fname := GrFname;
    ret := sio_FtYmodemRx(Port, fname, 1, rCallBack, 27);
end;
FTKERMIT:
begin
    fname := GrFname;
    ret := sio_FtKermitRx(Port, fname, 1, rCallBack, 27);
end;
FTASCII:
    ret := sio_FtASCIIRx(Port, GrFname, rCallBack, 27, 3);
end;
end;

if ret < 0 then { maybe something error }
    ProcessRet(port, ret, GProtocol, GDirection)
else
    if (GDirection = FT_XMIT) then
        Application.MessageBox(PChar('File Transmit OK'), PChar(GszAppName), MB_OK)
    else
        Application.MessageBox(PChar('File Receive OK'), PChar(GszAppName), MB_OK);
        FTTransForm.UpdateFT;
end;

function xCallBack(xmitlen: LongInt; buflen: LongInt; buf: PChar; flen: LongInt):
    LongInt; stdcall;
begin
    if GftCancel then
        begin
            xCallBack := -1; { this will terminate file transfer }
            Exit;
        end;

    FTTransForm.RefreshDlg(xmitlen, flen, GxFname);
    xCallBack := 0;
end;

function rCallBack(recvlen: LongInt; buflen: LongInt; buf: PChar; flen: LongInt):
    LongInt; stdcall;

```

```

begin
if GftCancel then
begin
rCallBack := -1; { this will terminate file transfer }
Exit;
end;

FTTransForm.RefreshDlg(recvlen, flen, GrFname);
rCallBack := 0;
end;

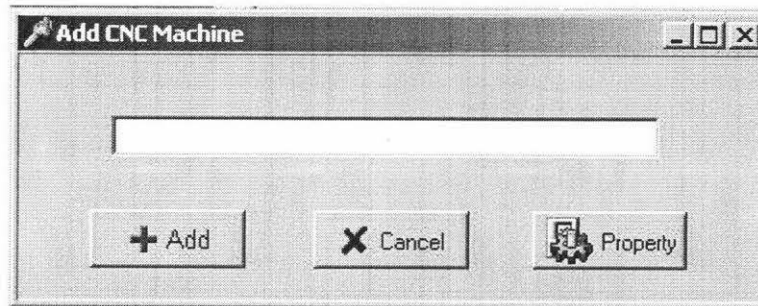
procedure ProcessRet(port:LongInt;ret:LongInt;protocol:Word;direction:Word);
var
buf : string;
begin
if (ret <> SIOFT_WIN32FAIL) then
begin
case ret of
SIOFT_BADPORT:
buf := 'Port is not opened in advance';
SIOFT_TIMEOUT:
if (direction = FT_RECV) then
buf := 'Receive timeout'
else
buf := 'Transmit Timeout';
SIOFT_FUNC:
if ((protocol = FTASCII) And (direction = FT_RECV)) then
{ When downloading ASCII file,user must press "Cancel"
button to stop ASCII receive }
buf := 'Receive File Ok'
else
buf := 'User abort';
SIOFT_FOPEN:
buf := 'Can"t open file';
SIOFT_CANABORT:
buf := 'Remote side abort';
SIOFT_BOARDNOTSUPPORT:
buf := 'Board does not support this function';

```

```
SIOFT_PROTOCOL, SIOFT_SKIP:
    buf := 'File transfer error';
else
    buf := 'File transfer error';
end;
Application.MessageBox(PChar(buf),PChar(GszAppName),MB_OK);
end
else
    ShowSysErr(GszAppName);
end;

end.
```

8. MACHFORM.PAS



```
unit MachForm;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  
StdCtrls, Buttons, ExtCtrls, FTransM;
```

```
type
```

```
TMachineForm = class(TForm)
```

```
  Bevel1: TBevel;
```

```
  EditNew: TEdit;
```

```
  AddBtn: TBitBtn;
```

```
  CancelBtn: TBitBtn;
```

```
  BitBtn1: TBitBtn;
```

```
  procedure AddBtnClick(Sender: TObject);
```

```
  procedure CancelBtnClick(Sender: TObject);
```

```
  procedure BitBtn1Click(Sender: TObject);
```

```
  procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
```

```
private
```

```
  { Private declarations }
```

```
public
```

```
  { Public declarations }
```

```
end;
```

```
var
```

```
  MachineForm: TMachineForm;
```


implementation

uses Prop;

{\$R *.DFM}

```
procedure TMachineForm.AddBtnClick(Sender: TObject);
begin
    if (EditNew.Text <> "") and
        (FTransForm.OutLine1.Lines.IndexOf (EditNew.Text) < 0) then
        begin
            {add the string to both listboxes}
            FTransForm.OutLine1.Lines.Add (EditNew.Text);
            Close;
        end;
end;

procedure TMachineForm.CancelBtnClick(Sender: TObject);
begin
    Close;
end;

procedure TMachineForm.BitBtn1Click(Sender: TObject);
begin
    PropertyForm.ShowModal;
end;

procedure TMachineForm.FormCloseQuery(Sender: TObject;
    var CanClose: Boolean);
begin
    EditNew.Text := ""
end;

end.
```

9. MXTOOL.PAS

```

(*****
MxTool.pas
-- Process PComm function return value

*****)

unit MxTool;

interface

procedure ShowSysErr(title:string);
procedure MxShowError(title:string;errcode:LongInt);

implementation

uses
  Windows,Dialogs,PComm,SysUtils,Forms;

procedure MxShowError(title:string;errcode:LongInt);
var
  buf:string;
begin
  if errcode <> SIO_WIN32FAIL then
  begin
    case errcode of
      SIO_BADPORT:
        buf := 'Port number is invalid or port is not opened in advance';
      SIO_OUTCONTROL:
        buf := 'This board does not support this function';
      SIO_NODATA:
        buf := 'No data to read';
      SIO_OPENFAIL:
        buf := 'No such port or port is occupied by other program';
      SIO_RTS_BY_HW:
        buf := 'RTS can"t be set because H/W flowctrl';
      SIO_BADPARM:

```

```

    buf := 'Bad parameter';
SIO_BOARDNOTSUPPORT:
    buf := 'This board does not support this function';
SIO_ABORT_WRITE:
    buf := 'Write has blocked, and user abort write';
SIO_WRITETIMEOUT:
    buf := 'Write timeout has happened';
else
    buf := 'Unknown Error:'+IntToStr(errcode);
end;
Application.MessageBox(PChar(buf),PChar(title),MB_OK or MB_ICONSTOP);
end
else
    ShowSysErr(title);
end;

procedure ShowSysErr(title:string);
var
    syserr :LongInt;
    lpMsgBuf:array[0..79] of Char;
    lang   :LongInt;
begin
    syserr := GetLastError();

    {MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT) }
    lang := (SUBLANG_DEFAULT shl 10) + LANG_NEUTRAL;

    FormatMessage(
        FORMAT_MESSAGE_FROM_SYSTEM,
        nil,
        syserr,
        lang,
        @lpMsgBuf,
        80,
        nil
    );
    Application.MessageBox(lpMsgBuf,PChar(title),MB_OK or MB_ICONSTOP);
end; end.

```

10. PCOMM.PAS

(*****

PComm.pas

-- PComm Lib unit for Delphi (32 bit version).

*****)

unit PComm;

interface

const

{ baud rate setting }

B50 = \$0;

B75 = \$1;

B110 = \$2;

B134 = \$3;

B150 = \$4;

B300 = \$5;

B600 = \$6;

B1200 = \$7;

B1800 = \$8;

B2400 = \$9;

B4800 = \$A;

B7200 = \$B;

B9600 = \$C;

B19200 = \$D;

B38400 = \$E;

B57600 = \$F;

B115200 = \$10;

B230400 = \$11;

B460800 = \$12;

B921600 = \$13;

{ data bit }

BIT_5 = \$0;

BIT_6 = \$1;

```
BIT_7 = $2;
```

```
BIT_8 = $3;
```

```
{ stop bit }
```

```
STOP_1 = $0;
```

```
STOP_2 = $4;
```

```
{ parity }
```

```
P_EVEN = $18;
```

```
P_ODD = $8;
```

```
P_SPC = $38;
```

```
P_MRK = $28;
```

```
P_NONE = $0;
```

```
{ modem control setting }
```

```
C_DTR = $1;
```

```
C_RTS = $2;
```

```
{ modem line status }
```

```
S_CTS = $1;
```

```
S_DSR = $2;
```

```
S_RI = $4;
```

```
S_CD = $8;
```

```
{ error code }
```

```
SIO_OK = 0;
```

```
SIO_BADPORT = -1; { No such port or port not opened }
```

```
SIO_OUTCONTROL = -2; { Can't control board }
```

```
SIO_NODATA = -4; { No data to read or no buffer to write }
```

```
SIO_OPENFAIL = -5; { No such port or port has opened }
```

```
SIO_RTS_BY_HW = -6; { Can't set because H/W flowctrl }
```

```
SIO_BADPARAM = -7; { Bad parameter }
```

```
SIO_WIN32FAIL = -8; (* Call win32 function fail, please call }
```

```
GetLastError to get the error code *)
```

```
SIO_BOARDNOTSUPPORT = -9; { Board does not support this function}
```

```
SIO_FAIL = -10; { PComm function run result fail }
```

```
SIO_ABORT_WRITE = -11; { Write has blocked, and user abort write }
```

```
SIO_WRITETIMEOUT = -12; { Write timeout has happened }
```

```

{ file transfer error code }
SIOFT_OK      = 0;
SIOFT_BADPORT = -1;  { No such port or port not open }
SIOFT_TIMEOUT = -2; { Protocol timeout }
SIOFT_ABORT   = -3;  { User key abort }
SIOFT_FUNC    = -4;  { Func return abort }
SIOFT_FOPEN   = -5;  { Can not open files }
SIOFT_CANABORT = -6; { Ymodem CAN signal abort }
SIOFT_PROTOCOL = -7; { Protocol checking error abort }
SIOFT_SKIP    = -8; { Zmodem remote skip this send file }
SIOFT_LACKRBUF = -9;  { Zmodem Recv-Buff size must >= 2K bytes }
SIOFT_WIN32FAIL = -10; (* OS fail }
                    GetLastError to get the error code *)
SIOFT_BOARDNOTSUPPORT = -11; { Board does not support this function}

```

type

```
IrqProc = procedure(port: Longint);stdcall;
```

```
CallBackProc = function(len: Longint; rlen: Longint; buf: PChar; flen: Longint): Longint;stdcall;
```

```
{Import routine from PComm.dll}
```

```
function sio_open(port: Longint): Longint; stdcall;
```

```
function sio_close(port: Longint): Longint; stdcall;
```

```
function sio_ioctl(port, baud, mode: Longint): Longint; stdcall;
```

```
function sio_flowctrl(port, mode: Longint): Longint; stdcall;
```

```
function sio_flush(port, func: Longint): Longint; stdcall;
```

```
function sio_DTR(port, mode: Longint): Longint; stdcall;
```

```
function sio_RTS(port, mode: Longint): Longint; stdcall;
```

```
function sio_lctrl(port, mode: Longint): Longint; stdcall;
```

```
function sio_baud(port, speed: Longint): Longint; stdcall;
```

```
function sio_getch(port: Longint): Longint; stdcall;
```

```
function sio_read(port: Longint; buf: PChar; len: Longint): Longint; stdcall;
```

```
function sio_linput(port: Longint; buf:PChar; len: Longint; term:Longint): Longint; stdcall;
```

```
function sio_putch(port, term: Longint): Longint; stdcall;
```

```
function sio_putb(port: Longint; buf:PChar; len: Longint): Longint; stdcall;
```

```
function sio_write(port: Longint; buf:PChar; len: Longint): Longint; stdcall;
```

```
function sio_putb_x(port: Longint; buf:PChar; len: Longint; tick:Longint): Longint; stdcall;
```

```

function sio_putb_x_ex(port: Longint; buf:PChar; len: Longint; tms:Longint): Longint; stdcall;
function sio_lstatus(port: Longint): Longint; stdcall;
function sio_iqueue(port: Longint): Longint; stdcall;
function sio_oqueue(port: Longint): Longint; stdcall;
function sio_Tx_hold(port: Longint): Longint; stdcall;
function sio_getbaud(port: Longint): Longint; stdcall;
function sio_getmode(port: Longint): Longint; stdcall;
function sio_getflow(port: Longint): Longint; stdcall;
function sio_data_status(port: Longint): Longint; stdcall;
function sio_term_irq(port: Longint; func: IrqProc; code: Byte): Longint; stdcall;
function sio_cnt_irq(port: Longint; func: IrqProc; count: Longint): Longint; stdcall;
function sio_modem_irq(port: Longint; func: IrqProc): Longint; stdcall;
function sio_break_irq(port: Longint; func: IrqProc): Longint; stdcall;
function sio_Tx_empty_irq(port: Longint; func: IrqProc): Longint; stdcall;
function sio_break(port, time: Longint): Longint; stdcall;
function sio_view(port: Longint; buf: PChar; len: Longint): Longint; stdcall;
function sio_TxLowWater(port, size: Longint): Longint; stdcall;
function sio_AbortWrite(port: Longint): Longint; stdcall;
function sio_AbortRead(port: Longint): Longint; stdcall;
function sio_SetWriteTimeouts(port, timeouts: Longint): Longint; stdcall;
function sio_GetWriteTimeouts(port: Longint; var TotalTimeouts:Longint): Longint; stdcall;
function sio_SetReadTimeouts(port, TotalTimeouts, IntervalTimeouts: Longint): Longint; stdcall;
function sio_GetReadTimeouts(port: Longint; var TotalTimeouts, IntervalTimeouts: Longint):
Longint; stdcall;
function sio_FtASCIITx(port:Longint; fname:PChar; func:CallBackProc; key:Longint): Longint;
stdcall;
function sio_FtASCIIRx(port:Longint; fname:PChar; func:CallBackProc; key:Longint; sec:Longint):
Longint; stdcall;
function sio_FtXmodemChecksumTx(port:Longint; fname:PChar; func:CallBackProc;
key:Longint): Longint; stdcall;
function sio_FtXmodemChecksumRx(port:Longint; fname:PChar; func:CallBackProc;
key:Longint): Longint; stdcall;
function sio_FtXmodemCRCTx(port:Longint; fname:PChar; func:CallBackProc; key:Longint):
Longint; stdcall;
function sio_FtXmodemCRCRx(port:Longint; fname:PChar; func:CallBackProc; key:Longint):
Longint; stdcall;
function sio_FtXmodem1KCRCTx(port:Longint; fname:PChar; func:CallBackProc; key:Longint):
Longint; stdcall;

```

```

function sio_FtXmodem1KCRCRx(port:Longint; fname:PChar; func:CallBackProc; key:Longint):
Longint; stdcall;
function sio_FtYmodemTx(port:Longint; fname:PChar; func:CallBackProc; key:Longint): Longint;
stdcall;
function sio_FtYmodemRx(port:Longint; var fname:PChar;fno:LongInt;func:CallBackProc;
key:Longint): Longint; stdcall;
function sio_FtZmodemTx(port:Longint; fname:PChar; func:CallBackProc; key:Longint): Longint;
stdcall;
function sio_FtZmodemRx(port:Longint; var fname:PChar;fno:LongInt;func:CallBackProc;
key:Longint): Longint; stdcall;
function sio_FtKermitTx(port:Longint; fname:PChar; func:CallBackProc; key:Longint): Longint;
stdcall;
function sio_FtKermitRx(port:Longint; var fname:PChar;fno:LongInt;func:CallBackProc;
key:Longint): Longint; stdcall;

```

implementation

```

function sio_open; external 'PComm.dll';
function sio_close; external 'PComm.dll';
function sio_ioctl; external 'PComm.dll';
function sio_flowctrl; external 'PComm.dll';
function sio_flush; external 'PComm.dll';
function sio_DTR; external 'PComm.dll';
function sio_RTS; external 'PComm.dll';
function sio_lctrl; external 'PComm.dll';
function sio_baud; external 'PComm.dll';
function sio_getch; external 'PComm.dll';
function sio_read; external 'PComm.dll';
function sio_input; external 'PComm.dll';
function sio_putch; external 'PComm.dll';
function sio_putb; external 'PComm.dll';
function sio_write; external 'PComm.dll';
function sio_putb_x; external 'PComm.dll';
function sio_putb_x_ex; external 'PComm.dll';
function sio_lstatus; external 'PComm.dll';
function sio_iqueue; external 'PComm.dll';
function sio_oqueue; external 'PComm.dll';
function sio_Tx_hold; external 'PComm.dll';

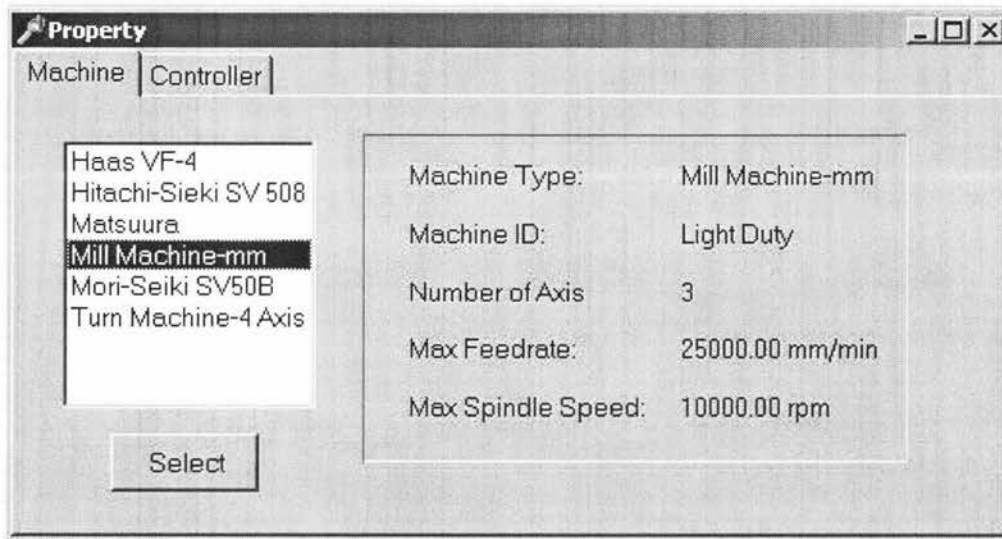
```



```
function sio_getbaud; external 'PComm.dll';
function sio_getmode; external 'PComm.dll';
function sio_getflow; external 'PComm.dll';
function sio_data_status; external 'PComm.dll';
function sio_term_irq; external 'PComm.dll';
function sio_cnt_irq; external 'PComm.dll';
function sio_modem_irq; external 'PComm.dll';
function sio_break_irq; external 'PComm.dll';
function sio_Tx_empty_irq; external 'PComm.dll';
function sio_break; external 'PComm.dll';
function sio_view; external 'PComm.dll';
function sio_TxLowWater; external 'PComm.dll';
function sio_AbortWrite; external 'PComm.dll';
function sio_AbortRead; external 'PComm.dll';
function sio_SetWriteTimeouts; external 'PComm.dll';
function sio_GetWriteTimeouts; external 'PComm.dll';
function sio_SetReadTimeouts; external 'PComm.dll';
function sio_GetReadTimeouts; external 'PComm.dll';
function sio_FtASCIITx; external 'PComm.dll';
function sio_FtASCIIRx; external 'PComm.dll';
function sio_FtXmodemCheckSumTx; external 'PComm.dll';
function sio_FtXmodemCheckSumRx; external 'PComm.dll';
function sio_FtXmodemCRCTx; external 'PComm.dll';
function sio_FtXmodemCRCRx; external 'PComm.dll';
function sio_FtXmodem1KCRCTx; external 'PComm.dll';
function sio_FtXmodem1KCRCRx; external 'PComm.dll';
function sio_FtYmodemTx; external 'PComm.dll';
function sio_FtYmodemRx; external 'PComm.dll';
function sio_FtZmodemTx; external 'PComm.dll';
function sio_FtZmodemRx; external 'PComm.dll';
function sio_FtKermitTx; external 'PComm.dll';
function sio_FtKermitRx; external 'PComm.dll';
```

end.

11. PROP.PAS



```
unit Prop;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  
StdCtrls, ExtCtrls, Buttons, ComCtrls;
```

```
type
```

```
TPropertyForm = class(TForm)  
  Properties: TPageControl;  
  Machine: TTabSheet;  
  Controller: TTabSheet;  
  ListBox2: TListBox;  
  Select: TBitBtn;  
  Bevel1: TBevel;  
  MachName: TLabel;  
  ContType: TLabel;  
  ZHOME: TLabel;  
  TravRate: TLabel;  
  ContLabel: TLabel;
```

```
MachineLabel: TLabel;  
ZHOMELabel: TLabel;  
TRAVRATELabel: TLabel;  
ListBox1: TListBox;  
BitBtn1: TBitBtn;  
MachType: TLabel;  
MaxFeed: TLabel;  
MachTypeLabel: TLabel;  
MachDutyLabel: TLabel;  
NoAxisLabel: TLabel;  
MaxFeedLabel: TLabel;  
Bevel2: TBevel;  
NoAxis: TLabel;  
MaxSpinSpeed: TLabel;  
MachineDuty: TLabel;  
MaxSpinLabel: TLabel;  
  
procedure SelectClick(Sender: TObject);  
procedure ListBox1Click(Sender: TObject);  
  
private  
  { Private declarations }  
public  
  { Public declarations }  
end;  
  
var  
  PropertyForm: TPropertyForm;  
  
implementation  
  
{$R *.DFM}  
  
procedure TPropertyForm.ListBox1Click(Sender: TObject);  
var  
  ListItem: Integer;  
begin
```

```
{look at each item of the multiple selection listbox}
for ListItem := 0 to ListBox1.Items.Count - 1 do
  if ListBox1.Selected [ListItem] then
    begin
      if ListItem = 0 then
        begin
          MachTypeLabel.Caption := 'Haas VF-4 Mill';
          MachDutyLabel.Caption := 'Light Duty';
          NoAxisLabel.Caption := '4';
          MaxFeedLabel.Caption := '25000.00 mm/min';
          MaxSpinLabel.Caption := '10000.00 rpm';
        end;

      if ListItem = 1 then
        begin
          MachTypeLabel.Caption := 'Hitachi Sieki Mill';
          MachDutyLabel.Caption := 'Light Duty';
          NoAxisLabel.Caption := '3';
          MaxFeedLabel.Caption := '25000.00 mm/min';
          MaxSpinLabel.Caption := '10000.00 rpm';
        end;

      if ListItem = 2 then
        begin
          MachTypeLabel.Caption := 'Matsuura Mill';
          MachDutyLabel.Caption := 'Light Duty';
          NoAxisLabel.Caption := '3';
          MaxFeedLabel.Caption := '25000.00 mm/min';
          MaxSpinLabel.Caption := '10000.00 rpm';
        end;

      if ListItem = 3 then
        begin
          MachTypeLabel.Caption := 'Mill Machine-mm';
          MachDutyLabel.Caption := 'Light Duty';
          NoAxisLabel.Caption := '3';
          MaxFeedLabel.Caption := '25000.00 mm/min';
          MaxSpinLabel.Caption := '10000.00 rpm';
        end;
      end;
    end;
```

```

end;

if ListItem = 4 then
begin
    MachTypeLabel.Caption := 'Mori-Sieki SV Mill';
    MachDutyLabel.Caption := 'Light Duty';
    NoAxisLabel.Caption := '3';
    MaxFeedLabel.Caption := '25000.00 mm/min';
    MaxSpinLabel.Caption := '10000.00 rpm';
end;

end;

end;

procedure TPropertyForm.SelectClick(Sender: TObject);
var
    ListItem: Integer;
begin
    {look at each item of the multiple selection listbox}
    for ListItem := 0 to ListBox1.Items.Count - 1 do
        if ListBox2.Selected [ListItem] then
            begin
                if ListItem = 0 then
                    begin
                        ContLabel.Caption := 'ACROMATIC 2100';
                        MachineLabel.Caption := 'CINCINNATI';
                        ZHomeLabel.Caption := '508.00mm';
                        TRAVRATELabel.Caption := '6350.00mm';
                    end;

                if ListItem = 1 then
                    begin
                        ContLabel.Caption := 'ACROMATIC 850';
                        MachineLabel.Caption := 'CINCINNATI';
                        ZHomeLabel.Caption := '508.00mm';
                        TRAVRATELabel.Caption := '6350.00mm';
                    end;
                end;
            end;
        end;
    end;
end;

```

```
if ListItem = 2 then
begin
    ContLabel.Caption := 'FADAL CNC 88';
    MachineLabel.Caption := 'FADAL';
    ZHomeLabel.Caption := '508.00mm';
    TRAVRATELabel.Caption := '6350.00mm';
end;
```

```
if ListItem = 2 then
begin
    ContLabel.Caption := 'MORI SEIKI';
    MachineLabel.Caption := 'FANUC 3000C';
    ZHomeLabel.Caption := '508.00mm';
    TRAVRATELabel.Caption := '6350.00mm';
end;
```

```
if ListItem = 3 then
begin
    ContLabel.Caption := 'MILL TUTORIAL';
    MachineLabel.Caption := 'FANUC TYPE';
    ZHomeLabel.Caption := '508.00mm';
    TRAVRATELabel.Caption := '250.00mm';
end;
```

```
if ListItem = 4 then
begin
    ContLabel.Caption := 'MILL TUTORIAL';
    MachineLabel.Caption := 'FANUC TYPE';
    ZHomeLabel.Caption := '508.00mm';
    TRAVRATELabel.Caption := '250.00mm';
end;
```

```
if ListItem = 5 then
begin
    ContLabel.Caption := 'HAAS CONTROL';
    MachineLabel.Caption := 'HAAS';
    ZHomeLabel.Caption := '508.00mm';
end;
```

```
    TRAVRATELabel.Caption := '6350.00mm';  
end;  
  
if ListItem = 6 then  
begin  
    ContLabel.Caption := 'HEIDENHAIN TNC 145';  
    MachineLabel.Caption := 'WELLS INDEX';  
    ZHomeLabel.Caption := '508.00mm';  
    TRAVRATELabel.Caption := '6350.00mm';  
end;  
end;  
end;  
  
end.
```

12. READTHD.PAS

```
(*****
ReadThd.pas
-- Read Thread for example program.
*****)
```

```
unit ReadThd;
```

```
interface
```

```
uses Classes;
```

```
type
```

```
TReadThread = class(TThread)
```

```
private
```

```
m_buf : array [0..511] of Char;
```

```
{ Private declarations }
```

```
protected
```

```
procedure Execute; override;
```

```
procedure ShowData;
```

```
end;
```

```
implementation
```

```
uses Windows, PComm, ExGlobal, FtransM;
```

```
{ TReadThread }
```

```
procedure TReadThread.ShowData;
```

```
var
```

```
lend : LongInt;
```

```
begin
```

```
(*
```

```
When got any data,dump buffer to Edit window.
```


NOTE:

If any Null character in buffer,
characters after null can't be dumped
to Edit window.

*)

```
lend := Length(GhForm.Term.Text);
```

```
if(lend>25000)then
```

```
begin
```

```
  { Edit Control buffer size limit }
```

```
  GhForm.Term.Text := string(m_buf);
```

```
  Exit;
```

```
end;
```

```
if(lend>25000)then
```

```
  GhForm.Term.SelStart := lend;
```

```
  GhForm.Term.SelLength := 0;
```

```
  GhForm.Term.SelText := string(m_buf);
```

```
end;
```

```
procedure TReadThread.Execute;
```

```
var
```

```
  len : LongInt;
```

```
begin
```

```
  (* before close port,set GhExit to true to terminate  
  the read thread *)
```

```
  while not GhExit do
```

```
    begin
```

```
      Sleep(10);
```

```
      len := sio_read(GCommData.Port,@m_buf,511);
```

```
      if (len>0) then
```

```
        begin
```

```
          m_buf[len] := Char(0);{null terminated string}
```

```
          Synchronize(ShowData);
```

```
        end
```

```
    end;
```

```
end;
```

```
end.
```

13. SIMPLEM.PAS



```

(*****
  SimpleM.pas
  -- Main window for simple dumb terminal example program.

*****)

unit SimpleM;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, ExGlobal, StdCtrls, Buttons, ExtCtrls, ToolWin, ComCtrls;

type
  TSimpleForm = class(TExampleForm)
    Panel1: TPanel;
    SendButton: TSpeedButton;

    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  end;

```

```
procedure TermKeyPress(Sender: TObject; var Key: Char);
procedure ClosePort;
procedure cmClearClick(Sender: TObject);
function OpenPort:Boolean;
function PortSet:boolean;
procedure SendButtonClick(Sender: TObject);

private
  Modified: Boolean;
public

end;

var
  SimpleForm : TSimpleForm;

implementation

uses PComm,MxTool,Config,ReadThd, About, FTTransM;

{$R *.DFM}

procedure TSimpleForm.FormCreate(Sender: TObject);
begin
  GszAppName := 'Simple Demo';
  Term.Enabled := False;
  GbOpen := false;
  GhForm := SimpleForm;
end;

function TSimpleForm.PortSet:boolean;
var
  port : LongInt;
  mode : LongInt;
  hw,sw : LongInt;
  ret : LongInt;
begin
  port := GCommData.Port;
```

```
mode := GCommData.Parity or GCommData.ByteSize or GCommData.StopBits;  
PortSet := false;
```

```
if GCommData.Hw then  
  hw := 3    { bit0 and bit1 }  
else  
  hw := 0;
```

```
if GCommData.Sw then  
  sw := 12   { bit2 and bit3 }  
else  
  sw := 0;
```

```
ret := sio_ioctl(port,GCommData.BaudRate,mode);  
if ret<>SIO_OK then  
begin  
  MxShowError('sio_ioctl',ret);  
  Exit;  
end;
```

```
ret := sio_flowctrl(port,hw or sw);  
if ret<>SIO_OK then  
begin  
  MxShowError('sio_flowctrl',ret);  
  Exit;  
end;
```

```
ret := sio_DTR(port,Integer(GCommData.Dtr));  
if ret<>SIO_OK then  
begin  
  MxShowError('sio_DTR',ret);  
  Exit;  
end;
```

```
if not GCommData.Hw then  
begin  
  ret := sio_RTS(port,Integer(GCommData.Rts));  
  if ret<>SIO_OK then
```

```
begin
  MxShowError('sio_RTS',ret);
  Exit;
end;
end;

ShowStatus();
PortSet := True;
end;

function TSimpleForm.OpenPort:Boolean;
var
  ret:Integer;
begin
  OpenPort := false;
  ret := sio_open(GCommData.Port);
  if ret <> SIO_OK then
    begin
      MxShowError('sio_open',ret);
      Exit;
    end;

  if PortSet() = false then
    begin
      sio_close(GCommData.Port);
      Exit;
    end;
  OpenPort := true;
  GhExit := false;
  TReadThread.Create(false);
  GbOpen := true;
  ShowStatus();
end;

procedure TSimpleForm.ClosePort;
begin
```

```
GhExit := true;
sio_close (GCommData.Port);
GbOpen := False;
ShowStatus();
end;

procedure TSimpleForm.TermKeyPress(Sender: TObject; var Key: Char);
begin
    sio_putch(GCommData.Port,Integer(Key));

    // Key:=Char(0);
end;

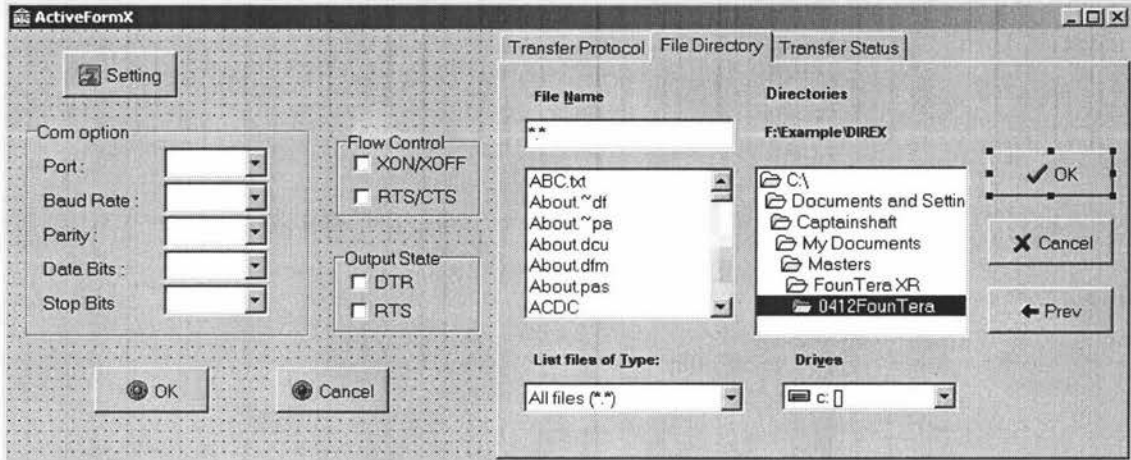
procedure TSimpleForm.cmClearClick(Sender: TObject);
begin
    Term.Clear();
end;

procedure TSimpleForm.SendButtonClick(Sender: TObject);
begin
    Term.Enabled :=True;
end;

procedure TSimpleForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    FtransForm.PortClose.Click;
    Close;
end;

end.
```

14. ACTIVEFORMIMPL1



```
unit ActiveFormImpl1;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  
ActiveX, AxCtrls, ActiveFormProj1_TLB, StdCtrls, Buttons, ExtCtrls,  
FileCtrl, ComCtrls;
```

```
type
```

```
TActiveFormX = class(TActiveForm, IActiveFormX)  
  GroupBox2: TGroupBox;  
  chHw: TCheckBox;  
  chSw: TCheckBox;  
  GroupBox1: TGroupBox;  
  Label1: TLabel;  
  Label2: TLabel;  
  Label3: TLabel;  
  Label4: TLabel;  
  Label5: TLabel;  
  cbPort: TComboBox;  
  cbBaudRate: TComboBox;
```

cbParity: TComboBox;
cbByteSize: TComboBox;
cbStopBits: TComboBox;
OK: TBitBtn;
Cancel: TBitBtn;
GroupBox3: TGroupBox;
chDtr: TCheckBox;
chRts: TCheckBox;
cmSetting: TBitBtn;
PageControl1: TPageControl;
TabSheet1: TTabSheet;
rgProtocol: TRadioGroup;
rgDirection: TRadioGroup;
FSetOk: TBitBtn;
FSetCancel: TBitBtn;
TabSheet2: TTabSheet;
Label6: TLabel;
ListFilesLabel: TLabel;
DrivesLabel: TLabel;
DirLabel: TLabel;
FileNameLabel: TLabel;
DirBox: TDirectoryListBox;
DirDlgOK: TBitBtn;
DirDlgCancel: TBitBtn;
DriveBox: TDriveComboBox;
DirDlgPrev: TBitBtn;
FileListBox: TFileListBox;
FileNameEdit: TEdit;
FilterComboBox: TFilterComboBox;
TabSheet3: TTabSheet;
TPort: TLabel;
TFileSize: TLabel;
TProtocol: TLabel;
TLength: TLabel;
TFileName: TLabel;
Bevel1: TBevel;
Bevel2: TBevel;
Bevel3: TBevel;


```
Bevel4: TBevel;
Bevel5: TBevel;
lbFname: TLabel;
lbFSize: TLabel;
lbxLen: TLabel;
lbPort: TLabel;
lbProtocol: TLabel;
TCancel: TBitBtn;
procedure FormCreate(Sender: TObject);
procedure chHwClick(Sender: TObject);
procedure OKClick(Sender: TObject);
procedure cmSettingClick(Sender: TObject);
function OpenPort:Boolean;
function PortSet:Boolean;
procedure FSetOkClick(Sender: TObject);
procedure CancelClick(Sender: TObject);
procedure ClosePort;
procedure FileListBoxDbClick(Sender: TObject);
procedure XmitFile;
procedure RecvFile;
procedure FSetCancelClick(Sender: TObject);
procedure DirDlgCancelClick(Sender: TObject);
procedure DirDlgOKClick(Sender: TObject);

private
{ Private declarations }
DoTra: Boolean;
FEvents: IActiveFormXEvents;
procedure ActivateEvent(Sender: TObject);
procedure ClickEvent(Sender: TObject);
procedure CreateEvent(Sender: TObject);
procedure DbClickEvent(Sender: TObject);
procedure DeactivateEvent(Sender: TObject);
procedure DestroyEvent(Sender: TObject);
procedure KeyPressEvent(Sender: TObject; var Key: Char);
procedure PaintEvent(Sender: TObject);
procedure UpDateFT;
procedure UpDateGT;
```

```
procedure UpDateHt;
procedure UpDateKt;
protected
{ Protected declarations }
procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage); override;
procedure EventSinkChanged(const EventSink: IUnknown); override;
function Get_Active: WordBool; safecall;
function Get_AutoScroll: WordBool; safecall;
function Get_AutoSize: WordBool; safecall;
function Get_AxBorderStyle: TxActiveFormBorderStyle; safecall;
function Get_BiDiMode: TxBiDiMode; safecall;
function Get_Caption: WideString; safecall;
function Get_Color: OLE_COLOR; safecall;
function Get_Cursor: Smallint; safecall;
function Get_DoubleBuffered: WordBool; safecall;
function Get_DropTarget: WordBool; safecall;
function Get_Enabled: WordBool; safecall;
function Get_Font: IFontDisp; safecall;
function Get_HelpFile: WideString; safecall;
function Get_KeyPreview: WordBool; safecall;
function Get_PixelsPerInch: Integer; safecall;
function Get_PrintScale: TxPrintScale; safecall;
function Get_Scaled: WordBool; safecall;
function Get_Visible: WordBool; safecall;
procedure _Set_Font(const Value: IFontDisp); safecall;
procedure Set_AutoScroll(Value: WordBool); safecall;
procedure Set_AutoSize(Value: WordBool); safecall;
procedure Set_AxBorderStyle(Value: TxActiveFormBorderStyle); safecall;
procedure Set_BiDiMode(Value: TxBiDiMode); safecall;
procedure Set_Caption(const Value: WideString); safecall;
procedure Set_Color(Value: OLE_COLOR); safecall;
procedure Set_Cursor(Value: Smallint); safecall;
procedure Set_DoubleBuffered(Value: WordBool); safecall;
procedure Set_DropTarget(Value: WordBool); safecall;
procedure Set_Enabled(Value: WordBool); safecall;
procedure Set_Font(var Value: IFontDisp); safecall;
procedure Set_HelpFile(const Value: WideString); safecall;
procedure Set_KeyPreview(Value: WordBool); safecall;
```

```

procedure Set_PixelsPerInch(Value: Integer); safecall;
procedure Set_PrintScale(Value: TxPrintScale); safecall;
procedure Set_Scaled(Value: WordBool); safecall;
procedure Set_Visible(Value: WordBool); safecall;
public
  { Public declarations }
  procedure Initialize; override;
end;

var
  ActiveXForm: TActiveFormX;
  GProtocol : Word;
  GDirection : Word;
  GftCancel : boolean;
  Gfhw : boolean;
  Falcon : String;
  GxFname : array[0..MAX_PATH] of Char;
  GrFname : array[0..MAX_PATH] of Char;
  GrPath : array[0..MAX_PATH] of Char;

  GstrProtocol:array [0..6] of string = (
    'XModem-1KCRC','XModem-Checksum','XModem-CRC',
    'ZModem','YModem','Kermit','ASCII'
  );

implementation

uses ComObj, ComServ, ExGlobal, PComm, MxTool, FtPro;

{$R *.DFM}

{ TActiveFormX }

procedure TActiveFormX.DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage);
begin
  { Define property pages here. Property pages are defined by calling

```

```
DefinePropertyPage with the class id of the page. For example,  
  DefinePropertyPage(Class_ActiveFormXPage); }  
end;  
  
procedure TActiveFormX.EventSinkChanged(const EventSink: IUnknown);  
begin  
  FEvents := EventSink as IActiveFormXEvents;  
end;  
  
procedure TActiveFormX.Initialize;  
begin  
  inherited Initialize;  
  OnActivate := ActivateEvent;  
  OnClick := ClickEvent;  
  OnCreate := CreateEvent;  
  OnDbClick := DbClickEvent;  
  OnDeactivate := DeactivateEvent;  
  OnDestroy := DestroyEvent;  
  OnKeyPress := KeyPressEvent;  
  OnPaint := PaintEvent;  
end;  
  
function TActiveFormX.Get_Active: WordBool;  
begin  
  Result := Active;  
end;  
  
function TActiveFormX.Get_AutoScroll: WordBool;  
begin  
  Result := AutoScroll;  
end;  
  
function TActiveFormX.Get_AutoSize: WordBool;  
begin  
  Result := AutoSize;  
end;  
  
function TActiveFormX.Get_AxBorderStyle: TxActiveFormBorderStyle;
```

```
begin
  Result := Ord(AxBorderStyle);
end;

function TActiveFormX.Get_BiDiMode: TxBiDiMode;
begin
  Result := Ord(BiDiMode);
end;

function TActiveFormX.Get_Caption: WideString;
begin
  Result := WideString(Caption);
end;

function TActiveFormX.Get_Color: OLE_COLOR;
begin
  Result := OLE_COLOR(Color);
end;

function TActiveFormX.Get_Cursor: Smallint;
begin
  Result := Smallint(Cursor);
end;

function TActiveFormX.Get_DoubleBuffered: WordBool;
begin
  Result := DoubleBuffered;
end;

function TActiveFormX.Get_DropTarget: WordBool;
begin
  Result := DropTarget;
end;

function TActiveFormX.Get_Enabled: WordBool;
begin
  Result := Enabled;
end;
```

```
function TActiveFormX.Get_Font: IFontDisp;
begin
  GetOleFont(Font, Result);
end;

function TActiveFormX.Get_HelpFile: WideString;
begin
  Result := WideString(HelpFile);
end;

function TActiveFormX.Get_KeyPreview: WordBool;
begin
  Result := KeyPreview;
end;

function TActiveFormX.Get_PixelsPerInch: Integer;
begin
  Result := PixelsPerInch;
end;

function TActiveFormX.Get_PrintScale: TxPrintScale;
begin
  Result := Ord(PrintScale);
end;

function TActiveFormX.Get_Scaled: WordBool;
begin
  Result := Scaled;
end;

function TActiveFormX.Get_Visible: WordBool;
begin
  Result := Visible;
end;

procedure TActiveFormX._Set_Font(const Value: IFontDisp);
begin
```

```
    SetOleFont(Font, Value);  
end;
```

```
procedure TActiveFormX.Set_AutoScroll(Value: WordBool);  
begin  
    AutoScroll := Value;  
end;
```

```
procedure TActiveFormX.Set_AutoSize(Value: WordBool);  
begin  
    AutoSize := Value;  
end;
```

```
procedure TActiveFormX.Set_AxBorderStyle(Value: TxActiveFormBorderStyle);  
begin  
    AxBorderStyle := TActiveFormBorderStyle(Value);  
end;
```

```
procedure TActiveFormX.Set_BiDiMode(Value: TxBiDiMode);  
begin  
    BiDiMode := TxBiDiMode(Value);  
end;
```

```
procedure TActiveFormX.Set_Caption(const Value: WideString);  
begin  
    Caption := TCaption(Value);  
end;
```

```
procedure TActiveFormX.Set_Color(Value: OLE_COLOR);  
begin  
    Color := TColor(Value);  
end;
```

```
procedure TActiveFormX.Set_Cursor(Value: Smallint);  
begin  
    Cursor := TCursor(Value);  
end;
```

```
procedure TActiveFormX.Set_DoubleBuffered(Value: WordBool);
begin
  DoubleBuffered := Value;
end;
```

```
procedure TActiveFormX.Set_DropTarget(Value: WordBool);
begin
  DropTarget := Value;
end;
```

```
procedure TActiveFormX.Set_Enabled(Value: WordBool);
begin
  Enabled := Value;
end;
```

```
procedure TActiveFormX.Set_Font(var Value: IFontDisp);
begin
  SetOleFont(Font, Value);
end;
```

```
procedure TActiveFormX.Set_HelpFile(const Value: WideString);
begin
  HelpFile := String(Value);
end;
```

```
procedure TActiveFormX.Set_KeyPreview(Value: WordBool);
begin
  KeyPreview := Value;
end;
```

```
procedure TActiveFormX.Set_PixelsPerInch(Value: Integer);
begin
  PixelsPerInch := Value;
end;
```

```
procedure TActiveFormX.Set_PrintScale(Value: TxPrintScale);
begin
  PrintScale := TPrintScale(Value);
end;
```



```
end;
```

```
procedure TActiveFormX.Set_Scaled(Value: WordBool);  
begin  
  Scaled := Value;  
end;
```

```
procedure TActiveFormX.Set_Visible(Value: WordBool);  
begin  
  Visible := Value;  
end;
```

```
procedure TActiveFormX.ActivateEvent(Sender: TObject);  
begin  
  if FEvents <> nil then FEvents.OnActivate;  
end;
```

```
procedure TActiveFormX.ClickEvent(Sender: TObject);  
begin  
  if FEvents <> nil then FEvents.OnClick;  
end;
```

```
procedure TActiveFormX.CreateEvent(Sender: TObject);  
begin  
  if FEvents <> nil then FEvents.OnCreate;  
end;
```

```
procedure TActiveFormX.DbClickEvent(Sender: TObject);  
begin  
  if FEvents <> nil then FEvents.OnDbClick;  
end;
```

```
procedure TActiveFormX.DeactivateEvent(Sender: TObject);  
begin  
  if FEvents <> nil then FEvents.OnDeactivate;  
end;
```

```
procedure TActiveFormX.DestroyEvent(Sender: TObject);
```

```
begin
  if FEvents <> nil then FEvents.OnDestroy;
end;

procedure TActiveFormX.KeyPressEvent(Sender: TObject; var Key: Char);
var
  TempKey: Smallint;
begin
  TempKey := Smallint(Key);
  if FEvents <> nil then FEvents.OnKeyPress(TempKey);
  Key := Char(TempKey);
end;

procedure TActiveFormX.PaintEvent(Sender: TObject);
begin
  if FEvents <> nil then FEvents.OnPaint;
end;

procedure TActiveFormX.FormCreate(Sender: TObject);
var
  i:Word;
begin
  DoTra := false;
  UpDateFT;
  for i:=1 to 256 do
    cbPort.Items.Add('COM'+IntToStr(i));
  with GCommData do
  begin
    Port := 1;
    ibaudrate := 14;
    iparity := 0;
    ibytesize := 3;
    istopbits := 0;
    BaudRate := B38400;
    Parity := P_NONE;
    ByteSize := BIT_8;
    StopBits := STOP_1;
    Hw := false;
```

```

Sw := false;
Dtr := true;
Rts := true;
end;
DirBox.FileList := FileListBox;
{This display the current directory as the caption of a label control}
DirBox.DirLabel := DirLabel;
{The displays the current drive so that the directory list box auomatically
updates its tree}
DriveBox.DirList := DirBox;
{This assign the edit objects name to the FileListBox's FileEdit property}
FileListBox.FileEdit := FileNameEdit;
end;

```

```

procedure TActiveFormX.UpDateFT;

```

```

begin

```

```

    cbPort.Enabled := DoTra;
    cbBaudRate.Enabled := DoTra;
    cbParity.Enabled := DoTra;
    cbByteSize.Enabled := DoTra;
    cbStopBits.Enabled := DoTra;
    chSw.Enabled := DoTra;
    chHw.Enabled := DoTra;
    chDtr.Enabled := DoTra;
    chRts.Enabled := DoTra;
    OK.Enabled := DoTra;
    Cancel.Enabled := DoTra;
    PageControl1.Enabled := DoTra;
    TabSheet1.Enabled := DoTra;
    FSetOk.Enabled := DoTra;
    FSetCancel.Enabled := DoTra;
    DirDlgOK.Enabled := DoTra;
    DirDlgCancel.Enabled := DoTra;
    DirDlgPrev.Enabled := DoTra;
    TCancel.Enabled := DoTra;

```

```

end;

```

```

procedure TActiveFormX.chHwClick(Sender: TObject);

```

```
begin
  chRts.Enabled := Gfhw;
  Gfhw := Not Gfhw;
end;

procedure TActiveFormX.cmSettingClick(Sender: TObject);

begin
  UpDateGT;
  with GCommData do
  begin
    cbPort.ItemIndex := Port-1;
    cbBaudRate.ItemIndex := ibaudrate;
    cbParity.ItemIndex := iparity;
    cbByteSize.ItemIndex := iblytesize;
    cbStopBits.ItemIndex := istopbits;
    chHw.Checked := Hw;
    chSw.Checked := Sw;
    chRts.Checked := Rts;
    chDtr.Checked := Dtr;
    Gfhw := Hw;
    chRts.Enabled := not Gfhw;
    { disable com port setting when open }
    cbPort.Enabled := not GbOpen;
  end;
end;

procedure TActiveFormX.UpDateGT;
begin
  cmSetting.Enabled := not DoTra;
  cbPort.Enabled := not DoTra;
  cbBaudRate.Enabled :=not DoTra;
  cbParity.Enabled := not DoTra;
  cbByteSize.Enabled := not DoTra;
  cbStopBits.Enabled := not DoTra;
  chSw.Enabled := not DoTra;
  chHw.Enabled := not DoTra;
  chDtr.Enabled := not DoTra;
```

```

chRts.Enabled := not DoTra;
OK.Enabled := not DoTra;
Cancel.Enabled := not DoTra;
PageControl1.Enabled := DoTra;
TabSheet1.Enabled := DoTra;
FSetOk.Enabled := DoTra;
FSetCancel.Enabled := DoTra;
end;

procedure TActiveFormX.OKClick(Sender: TObject);
var
  W:Word;
begin
with GCommData do
begin
  Port := cbPort.ItemIndex + 1;
  ibaudrate := cbBaudRate.ItemIndex;
  iparity := cbParity.ItemIndex;
  ibratesize := cbByteSize.ItemIndex;
  istopbits := cbStopBits.ItemIndex;
  BaudRate := GBaudTable[ibaudrate];
  ByteSize := GByteSizeTable[ibratesize];
  Parity := GParityTable[iparity];
  StopBits := GStopBitsTable[istopbits];
  Hw := chHw.Checked;
  Sw := chSw.Checked;
  Rts := chRts.Checked;
  Dtr := chDtr.Checked;
end;

begin
W:=MessageDlg('Settings Correct?', mtConfirmation,[mbYes, mbNo], 0);
  case W of
  mrYes: begin
    OpenPort();
    PageControl1.ActivePage := TabSheet1;
  end;
  mrNo: modalresult := mrcancel;

```

```
    end;  
end;  
end;
```

```
function TActiveFormX.OpenPort:Boolean;
```

```
var
```

```
    ret:Integer;
```

```
begin
```

```
    OpenPort := false;
```

```
    ret := sio_open(GCommData.Port);
```

```
    if ret <> SIO_OK then
```

```
        begin
```

```
            MxShowError('sio_open',ret);
```

```
            Exit;
```

```
        end;
```

```
    if PortSet() = false then
```

```
        begin
```

```
            sio_close(GCommData.Port);
```

```
            Exit;
```

```
        end;
```

```
    OpenPort := true;
```

```
    GhExit := false;
```

```
    //TReadThread.Create(false);
```

```
    GbOpen := true;
```

```
    //SwitchMenu();
```

```
    UpDateHt;
```

```
end;
```

```
procedure TActiveFormX.UpDateHt;
```

```
begin
```

```
    cmSetting.Enabled := DoTra;
```

```
    cbPort.Enabled := DoTra;
```

```
    cbBaudRate.Enabled := DoTra;
```

```
    cbParity.Enabled := DoTra;
```

```
    cbByteSize.Enabled := DoTra;
```

```
    cbStopBits.Enabled := DoTra;
```

```
    chSw.Enabled := DoTra;
```

```
    chHw.Enabled := DoTra;
```

```
chDtr.Enabled := DoTra;
chRts.Enabled := DoTra;
OK.Enabled := DoTra;
Cancel.Enabled := not DoTra;
PageControl1.Enabled := not DoTra;
TabSheet1.Enabled := not DoTra;
FSetOk.Enabled := not DoTra;
FSetCancel.Enabled := not DoTra;
end;

procedure TActiveFormX.ClosePort;
begin
  sio_close (GCommData.Port);
  GbOpen := False;
end;

function TActiveFormX.PortSet:boolean;
var
  port : LongInt;
  mode : LongInt;
  hw,sw : LongInt;
  ret : LongInt;
begin
  port := GCommData.Port;
  mode := GCommData.Parity or GCommData.ByteSize or GCommData.StopBits;
  PortSet := false;

  if GCommData.Hw then
    hw := 3    { bit0 and bit1 }
  else
    hw := 0;

  if GCommData.Sw then
    sw := 12   { bit2 and bit3 }
  else
    sw := 0;

  ret := sio_ioctl(port,GCommData.BaudRate,mode);
```

```
if ret<>SIO_OK then
```

```
begin
```

```
  MxShowError('sio_ioctl',ret);
```

```
  Exit;
```

```
end;
```

```
ret := sio_flowctrl(port,hw or sw);
```

```
if ret<>SIO_OK then
```

```
begin
```

```
  MxShowError('sio_flowctrl',ret);
```

```
  Exit;
```

```
end;
```

```
ret := sio_DTR(port,Integer(GCommData.Dtr));
```

```
if ret<>SIO_OK then
```

```
begin
```

```
  MxShowError('sio_DTR',ret);
```

```
  Exit;
```

```
end;
```

```
if not GCommData.Hw then
```

```
begin
```

```
  ret := sio_RTS(port,Integer(GCommData.Rts));
```

```
  if ret<>SIO_OK then
```

```
    begin
```

```
      MxShowError('sio_RTS',ret);
```

```
      Exit;
```

```
    end;
```

```
end;
```

```
PortSet := True;
```

```
end;
```

```
procedure TActiveFormX.FSetOkClick(Sender: TObject);
```

```
begin
```

```
  //Declare which protocol to use.
```

```
  GProtocol := rgProtocol.ItemIndex;
```

```
  //Declare whether to send or to receive file.
```



```
GDirection := rgDirection.ItemIndex;
//Update of the buttons and sheets.
if rgDirection.ItemIndex = FT_XMIT then
//Access the the second page of Tab.
  UpDateKt
else
  UpDateKt;
end;

procedure TActiveFormX.UpDateKt;
begin
  PageControl1.ActivePage := TabSheet2;
  DirDlgOk.Enabled := not DoTra;
  DirDlgCancel.Enabled := not DoTra;
  DirDlgPrev.Enabled := not DoTra;
end;

procedure TActiveFormX.CancelClick(Sender: TObject);
begin
  ClosePort();
  UpDateGT;
end;

procedure TActiveFormX.FileListBoxDbClick(Sender: TObject);
begin
  if rgDirection.ItemIndex = FT_XMIT then
    XmitFile
  end;

procedure TActiveFormX.FSetCancelClick(Sender: TObject);
begin
  GftCancel :=true;
end;

procedure TActiveFormX.DirDlgCancelClick(Sender: TObject);
begin
  GftCancel :=true;
end;
```

```

procedure TActiveFormX.XmitFile;
begin
  {Declare Falcon using the FileNameEdit string}
  Falcon := FileNameEdit.Text;
  {Istrcpy copies the entire contents of one string into another string.
  Either string, instead of being a "real" string, can also be merely a pointer to a string instead.
  The target string must already have enough space to receive the source string's contents.
  The function also will copy a terminating null character into the target string}
  Istrcpy(GxFname,PChar(Falcon));
  {If user press 'Cancel' button which on status dialog,
  'GftCancel' flag will be set to true.This will let callback
  function to return -1 to terminate file transfer.}
  GftCancel := false;
  TFtProC.Create(false);
  PageControl1.ActivePage := TabSheet3;
end;

```

```

procedure TActiveFormX.RecvFile;
begin
  GftCancel := false;
  TFtProc.Create(false);
end;

```

```

procedure TActiveFormX.DirDlgOKClick(Sender: TObject);
begin
  Falcon := FileNameEdit.Text;
  {Declare Falcon using the FileNameEdit string}
  if rgDirection.ItemIndex = FT_XMIT then
    XmitFile
  else
    begin
      if (GProtocol=FTZMDM) or (GProtocol=FTYMMDM) or (GProtocol=FTKERMIT)then
        begin
          Istrcpy(GrPath,PChar(DirBox.Directory));
          SetCurrentDir(GrPath)
        end
    end

```

```

else
begin
Istrcpy(GrFname,PChar(Falcon));
end;
PageControl1.ActivePage :=TabSheet3;
RecvFile;
end;
{Istrcpy copies the entire contents of one string into another string.
Either string, instead of being a "real" string, can also be merely a pointer to a string instead.
The target string must already have enough space to receive the source string's contents.
The function also will copy a terminating null character into the target string}

{If user press 'Cancel' button which on status dialog,
'GftCancel' flag will be set to true.This will let callback
function to return -1 to terminate file transfer.}
GftCancel := false;
TFtProC.Create(false);
PageControl1.ActivePage := TabSheet3;
end;

initialization
TActiveFormFactory.Create(
  ComServer,
  TActiveFormControl,
  TActiveFormX,
  Class_ActiveFormX,
  1,
  ",
  OLEMISC_SIMPLEFRAME or OLEMISC_ACTSLIKELABEL,
  tmApartment);
end.

```

15. ACTIVEFORMPROJ1_TLB

```

unit ActiveFormProj1_TLB;

// ***** //
// WARNING //
// ----- //
// The types declared in this file were generated from data read from a //
// Type Library. If this type library is explicitly or indirectly (via //
// another type library referring to this type library) re-imported, or the //
// 'Refresh' command of the Type Library Editor activated while editing the //
// Type Library, the contents of this file will be regenerated and all //
// manual modifications will be lost. //
// ***** //

// PASTLWTR : $Revision: 1.11.1.63 $
// File generated on 10/26/01 11:38:09 PM from Type Library described below.

// ***** //
// Type Lib: C:\CD\ActiveFormProj1.tlb
// IID\LCID: {A70D614D-C9F6-11D5-91DF-0000E85EFBBA}0
// Helpfile:
// HelpString: ActiveFormProj1 Library
// Version: 1.0
// ***** //

interface

uses Windows, ActiveX, Classes, Graphics, OleCtrls, StdVCL;

// *****//
// GUIDS declared in the TypeLibrary. Following prefixes are used: //
// Type Libraries : LIBID_xxxx //
// CoClasses : CLASS_xxxx //
// DISPInterfaces : DIID_xxxx //
// Non-DISP interfaces: IID_xxxx //
// *****//

```

```

const
LIBID_ActiveFormProj1: TGUID = '{A70D614D-C9F6-11D5-91DF-0000E85EFBBA}';
IID_IActiveFormX: TGUID = '{A70D614E-C9F6-11D5-91DF-0000E85EFBBA}';
DIID_IActiveFormXEvents: TGUID = '{A70D6150-C9F6-11D5-91DF-0000E85EFBBA}';
CLASS_ActiveFormX: TGUID = '{A70D6152-C9F6-11D5-91DF-0000E85EFBBA}';

// *****//
// Declaration of Enumerations defined in Type Library      //
// *****//
// TxActiveFormBorderStyle constants
type
  TxActiveFormBorderStyle = TOleEnum;
const
  afbNone = $00000000;
  afbSingle = $00000001;
  afbSunken = $00000002;
  afbRaised = $00000003;

// TxPrintScale constants
type
  TxPrintScale = TOleEnum;
const
  poNone = $00000000;
  poProportional = $00000001;
  poPrintToFit = $00000002;

// TxMouseButton constants
type
  TxMouseButton = TOleEnum;
const
  mbLeft = $00000000;
  mbRight = $00000001;
  mbMiddle = $00000002;

// TxBiDiMode constants
type
  TxBiDiMode = TOleEnum;
const

```

```

bdLeftToRight = $00000000;
bdRightToLeft = $00000001;
bdRightToLeftNoAlign = $00000002;
bdRightToLeftReadingOnly = $00000003;

```

type

```

// *****//
// Forward declaration of interfaces defined in Type Library //
// *****//
IActiveFormX = interface;
IActiveFormXDisp = dispinterface;
IActiveFormXEvents = dispinterface;

// *****//
// Declaration of CoClasses defined in Type Library //
// (NOTE: Here we map each CoClass to its Default Interface) //
// *****//
ActiveFormX = IActiveFormX;

// *****//
// Interface: IActiveFormX
// Flags: (4416) Dual OleAutomation Dispatchable
// GUID: {A70D614E-C9F6-11D5-91DF-0000E85EFBBA}
// *****//
IActiveFormX = interface(IDispatch)
    [{A70D614E-C9F6-11D5-91DF-0000E85EFBBA}]
    function Get_Visible: WordBool; safecall;
    procedure Set_Visible(Value: WordBool); safecall;
    function Get_AutoScroll: WordBool; safecall;
    procedure Set_AutoScroll(Value: WordBool); safecall;
    function Get_AutoSize: WordBool; safecall;
    procedure Set_AutoSize(Value: WordBool); safecall;
    function Get_AxBorderStyle: TxActiveFormBorderStyle; safecall;
    procedure Set_AxBorderStyle(Value: TxActiveFormBorderStyle); safecall;
    function Get_Caption: WideString; safecall;
    procedure Set_Caption(const Value: WideString); safecall;

```

```

function Get_Color: OLE_COLOR; safecall;
procedure Set_Color(Value: OLE_COLOR); safecall;
function Get_Font: IFontDisp; safecall;
procedure _Set_Font(const Value: IFontDisp); safecall;
procedure Set_Font(var Value: IFontDisp); safecall;
function Get_KeyPreview: WordBool; safecall;
procedure Set_KeyPreview(Value: WordBool); safecall;
function Get_PixelsPerInch: Integer; safecall;
procedure Set_PixelsPerInch(Value: Integer); safecall;
function Get_PrintScale: TxPrintScale; safecall;
procedure Set_PrintScale(Value: TxPrintScale); safecall;
function Get_Scaled: WordBool; safecall;
procedure Set_Scaled(Value: WordBool); safecall;
function Get_Active: WordBool; safecall;
function Get_DropTarget: WordBool; safecall;
procedure Set_DropTarget(Value: WordBool); safecall;
function Get_HelpFile: WideString; safecall;
procedure Set_HelpFile(const Value: WideString); safecall;
function Get_DoubleBuffered: WordBool; safecall;
procedure Set_DoubleBuffered(Value: WordBool); safecall;
function Get_Enabled: WordBool; safecall;
procedure Set_Enabled(Value: WordBool); safecall;
function Get_BiDiMode: TxBiDiMode; safecall;
procedure Set_BiDiMode(Value: TxBiDiMode); safecall;
function Get_Cursor: Smallint; safecall;
procedure Set_Cursor(Value: Smallint); safecall;
property Visible: WordBool read Get_Visible write Set_Visible;
property AutoScroll: WordBool read Get_AutoScroll write Set_AutoScroll;
property AutoSize: WordBool read Get_AutoSize write Set_AutoSize;
property AxBorderStyle: TxActiveFormBorderStyle read Get_AxBorderStyle write
Set_AxBorderStyle;
property Caption: WideString read Get_Caption write Set_Caption;
property Color: OLE_COLOR read Get_Color write Set_Color;
property Font: IFontDisp read Get_Font write _Set_Font;
property KeyPreview: WordBool read Get_KeyPreview write Set_KeyPreview;
property PixelsPerInch: Integer read Get_PixelsPerInch write Set_PixelsPerInch;
property PrintScale: TxPrintScale read Get_PrintScale write Set_PrintScale;
property Scaled: WordBool read Get_Scaled write Set_Scaled;

```

```

property Active: WordBool read Get_Active;
property DropTarget: WordBool read Get_DropTarget write Set_DropTarget;
property HelpFile: WideString read Get_HelpFile write Set_HelpFile;
property DoubleBuffered: WordBool read Get_DoubleBuffered write Set_DoubleBuffered;
property Enabled: WordBool read Get_Enabled write Set_Enabled;
property BiDiMode: TxBiDiMode read Get_BiDiMode write Set_BiDiMode;
property Cursor: Smallint read Get_Cursor write Set_Cursor;
end;

// *****//
// Displntf: IActiveFormXDisp
// Flags: (4416) Dual OleAutomation Dispatchable
// GUID: {A70D614E-C9F6-11D5-91DF-0000E85EFBBA}
// *****//
IActiveFormXDisp = dispinterface
  [{A70D614E-C9F6-11D5-91DF-0000E85EFBBA}]
  property Visible: WordBool dispid 1;
  property AutoScroll: WordBool dispid 2;
  property AutoSize: WordBool dispid 3;
  property AxBorderStyle: TxActiveFormBorderStyle dispid 4;
  property Caption: WideString dispid -518;
  property Color: OLE_COLOR dispid -501;
  property Font: IFontDisp dispid -512;
  property KeyPreview: WordBool dispid 5;
  property PixelsPerInch: Integer dispid 6;
  property PrintScale: TxPrintScale dispid 7;
  property Scaled: WordBool dispid 8;
  property Active: WordBool readonly dispid 9;
  property DropTarget: WordBool dispid 10;
  property HelpFile: WideString dispid 11;
  property DoubleBuffered: WordBool dispid 12;
  property Enabled: WordBool dispid -514;
  property BiDiMode: TxBiDiMode dispid 13;
  property Cursor: Smallint dispid 14;
end;

// *****//
// Displntf: IActiveFormXEvents

```



```

// Flags: (0)
// GUID: {A70D6150-C9F6-11D5-91DF-0000E85EFBBA}
// *****//

IActiveFormXEvents = dispinterface
  [{A70D6150-C9F6-11D5-91DF-0000E85EFBBA}]
  procedure OnActivate; dispid 1;
  procedure OnClick; dispid 2;
  procedure OnCreate; dispid 3;
  procedure OnDbClick; dispid 4;
  procedure OnDestroy; dispid 5;
  procedure OnDeactivate; dispid 6;
  procedure OnKeyPress(var Key: Smallint); dispid 10;
  procedure OnPaint; dispid 15;
end;

// *****//

// OLE Control Proxy class declaration
// Control Name : TActiveFormX
// Help String : ActiveFormX Control
// Default Interface: IActiveFormX
// Def. Intf. DISP? : No
// Event Interface: IActiveFormXEvents
// TypeFlags : (34) CanCreate Control
// *****//

TActiveFormXOnKeyPress = procedure(Sender: TObject; var Key: Smallint) of object;

TActiveFormX = class(TOleControl)
private
  FOnActivate: TNotifyEvent;
  FOnClick: TNotifyEvent;
  FOnCreate: TNotifyEvent;
  FOnDbClick: TNotifyEvent;
  FOnDestroy: TNotifyEvent;
  FOnDeactivate: TNotifyEvent;
  FOnKeyPress: TActiveFormXOnKeyPress;
  FOnPaint: TNotifyEvent;
  FIntf: IActiveFormX;

```

```

function GetControlInterface: IActiveFormX;
protected
  procedure CreateControl;
  procedure InitControlData; override;
public
  property ControlInterface: IActiveFormX read GetControlInterface;
  property Visible: WordBool index 1 read GetWordBoolProp write SetWordBoolProp;
  property Active: WordBool index 9 read GetWordBoolProp;
  property DropTarget: WordBool index 10 read GetWordBoolProp write SetWordBoolProp;
  property HelpFile: WideString index 11 read GetWideStringProp write SetWideStringProp;
  property DoubleBuffered: WordBool index 12 read GetWordBoolProp write SetWordBoolProp;
  property Enabled: WordBool index -514 read GetWordBoolProp write SetWordBoolProp;
  property BiDiMode: TOleEnum index 13 read GetTOleEnumProp write SetTOleEnumProp;
published
  property AutoScroll: WordBool index 2 read GetWordBoolProp write SetWordBoolProp stored
False;
  property AutoSize: WordBool index 3 read GetWordBoolProp write SetWordBoolProp stored
False;
  property AxBorderStyle: TOleEnum index 4 read GetTOleEnumProp write SetTOleEnumProp
stored False;
  property Caption: WideString index -518 read GetWideStringProp write SetWideStringProp
stored False;
  property Color: TColor index -501 read GetTColorProp write SetTColorProp stored False;
  property Font: TFont index -512 read GetTFontProp write SetTFontProp stored False;
  property KeyPreview: WordBool index 5 read GetWordBoolProp write SetWordBoolProp stored
False;
  property PixelsPerInch: Integer index 6 read GetIntegerProp write SetIntegerProp stored
False;
  property PrintScale: TOleEnum index 7 read GetTOleEnumProp write SetTOleEnumProp
stored False;
  property Scaled: WordBool index 8 read GetWordBoolProp write SetWordBoolProp stored
False;
  property Cursor: Smallint index 14 read GetSmallintProp write SetSmallintProp stored
False;
  property OnActivate: TNotifyEvent read FOnActivate write FOnActivate;
  property OnClick: TNotifyEvent read FOnClick write FOnClick;
  property OnCreate: TNotifyEvent read FOnCreate write FOnCreate;
  property OnDbClick: TNotifyEvent read FOnDbClick write FOnDbClick;
  property OnDestroy: TNotifyEvent read FOnDestroy write FOnDestroy;
  property OnDeactivate: TNotifyEvent read FOnDeactivate write FOnDeactivate;

```

```

    property OnKeyPress: TActiveFormXOnKeyPress read FOnKeyPress write FOnKeyPress;
    property OnPaint: TNotifyEvent read FOnPaint write FOnPaint;
end;

procedure Register;

implementation

uses ComObj;

procedure TActiveFormX.InitControlData;
const
    CEventDisplDs: array [0..7] of DWORD = (
        $00000001, $00000002, $00000003, $00000004, $00000005, $00000006,
        $0000000A, $0000000F);
    CTFontIDs: array [0..0] of DWORD = (
        $FFFFFFE0);
    CControlData: TControlData = (
        ClassID: '{A70D6152-C9F6-11D5-91DF-0000E85EFBBA}';
        EventIID: '{A70D6150-C9F6-11D5-91DF-0000E85EFBBA}';
        EventCount: 8;
        EventDisplDs: @CEventDisplDs;
        LicenseKey: nil;
        Flags: $0000001D;
        Version: 300;
        FontCount: 1;
        FontIDs: @CTFontIDs);
begin
    ControlData := @CControlData;
end;

procedure TActiveFormX.CreateControl;

    procedure DoCreate;
    begin
        FIntf := IUnknown(OleObject) as IActiveFormX;
    end;

```

```
begin
  if Flntf = nil then DoCreate;
end;

function TActiveFormX.GetControlInterface: IActiveFormX;
begin
  CreateControl;
  Result := Flntf;
end;

procedure Register;
begin
  RegisterComponents('ActiveX',[TActiveFormX]);
end;

end.
```

16. ACTIVEFORMPROJ1.IDL

```
[
  uuid(A70D614D-C9F6-11D5-91DF-0000E85EFBBA),
  version(1.0),
  helpstring("ActiveFormProj1 Library"),
  control
]
library ActiveFormProj1
{

  importlib("stdole2.tlb");
  importlib("STDVCL40.DLL");

  [
    uuid(A70D614E-C9F6-11D5-91DF-0000E85EFBBA),
    version(1.0),
    helpstring("Dispatch interface for ActiveFormX Control"),
    dual,
    oleautomation
  ]
  interface IActiveFormX: IDispatch
  {
    [propget, id(0x00000001), hidden]
    HRESULT _stdcall Visible([out, retval] VARIANT_BOOL * Value );
    [propput, id(0x00000001), hidden]
    HRESULT _stdcall Visible([in] VARIANT_BOOL Value );
    [propget, id(0x00000002)]
    HRESULT _stdcall AutoScroll([out, retval] VARIANT_BOOL * Value );
    [propput, id(0x00000002)]
    HRESULT _stdcall AutoScroll([in] VARIANT_BOOL Value );
    [propget, id(0x00000003)]
    HRESULT _stdcall AutoSize([out, retval] VARIANT_BOOL * Value );
    [propput, id(0x00000003)]
    HRESULT _stdcall AutoSize([in] VARIANT_BOOL Value );
    [propget, id(0x00000004)]
    HRESULT _stdcall AxBorderStyle([out, retval] TxActiveFormBorderStyle * Value );
    [propput, id(0x00000004)]
```

```

HRESULT _stdcall AxBorderStyle([in] TxActiveFormBorderStyle Value );
[propget, id(0xFFFFFDFA)]
HRESULT _stdcall Caption([out, retval] BSTR * Value );
[propput, id(0xFFFFFDFA)]
HRESULT _stdcall Caption([in] BSTR Value );
[propget, id(0xFFFFE0B)]
HRESULT _stdcall Color([out, retval] OLE_COLOR * Value );
[propput, id(0xFFFFE0B)]
HRESULT _stdcall Color([in] OLE_COLOR Value );
[propget, id(0xFFFFE00)]
HRESULT _stdcall Font([out, retval] IFontDisp ** Value );
[propput, id(0xFFFFE00)]
HRESULT _stdcall Font([in] IFontDisp * Value );
[propputref, id(0xFFFFE00)]
HRESULT _stdcall Font([in, out] IFontDisp ** Value );
[propget, id(0x00000005)]
HRESULT _stdcall KeyPreview([out, retval] VARIANT_BOOL * Value );
[propput, id(0x00000005)]
HRESULT _stdcall KeyPreview([in] VARIANT_BOOL Value );
[propget, id(0x00000006)]
HRESULT _stdcall PixelsPerInch([out, retval] long * Value );
[propput, id(0x00000006)]
HRESULT _stdcall PixelsPerInch([in] long Value );
[propget, id(0x00000007)]
HRESULT _stdcall PrintScale([out, retval] TxPrintScale * Value );
[propput, id(0x00000007)]
HRESULT _stdcall PrintScale([in] TxPrintScale Value );
[propget, id(0x00000008)]
HRESULT _stdcall Scaled([out, retval] VARIANT_BOOL * Value );
[propput, id(0x00000008)]
HRESULT _stdcall Scaled([in] VARIANT_BOOL Value );
[propget, id(0x00000009), hidden]
HRESULT _stdcall Active([out, retval] VARIANT_BOOL * Value );
[propget, id(0x0000000A), hidden]
HRESULT _stdcall DropTarget([out, retval] VARIANT_BOOL * Value );
[propput, id(0x0000000A), hidden]
HRESULT _stdcall DropTarget([in] VARIANT_BOOL Value );
[propget, id(0x0000000B), hidden]

```

```

HRESULT _stdcall HelpFile([out, retval] BSTR * Value );
[propput, id(0x0000000B), hidden]
HRESULT _stdcall HelpFile([in] BSTR Value );
[propget, id(0x0000000C), hidden]
HRESULT _stdcall DoubleBuffered([out, retval] VARIANT_BOOL * Value );
[propput, id(0x0000000C), hidden]
HRESULT _stdcall DoubleBuffered([in] VARIANT_BOOL Value );
[propget, id(0xFFFFDFE), hidden]
HRESULT _stdcall Enabled([out, retval] VARIANT_BOOL * Value );
[propput, id(0xFFFFDFE), hidden]
HRESULT _stdcall Enabled([in] VARIANT_BOOL Value );
[propget, id(0x0000000D), hidden]
HRESULT _stdcall BiDiMode([out, retval] TxBiDiMode * Value );
[propput, id(0x0000000D), hidden]
HRESULT _stdcall BiDiMode([in] TxBiDiMode Value );
[propget, id(0x0000000E)]
HRESULT _stdcall Cursor([out, retval] short * Value );
[propput, id(0x0000000E)]
HRESULT _stdcall Cursor([in] short Value );
};

[
  uuid(A70D6150-C9F6-11D5-91DF-0000E85EFBBA),
  version(1.0),
  helpstring("Events interface for ActiveFormX Control")
]
dispinterface IActiveFormXEvents
{
  properties:
  methods:
  [id(0x00000001)]
  void OnActivate( void );
  [id(0x00000002)]
  void OnClick( void );
  [id(0x00000003)]
  void OnCreate( void );
  [id(0x00000004)]
  void OnDbClick( void );
}

```

```

[id(0x00000005)]
void OnDestroy( void );
[id(0x00000006)]
void OnDeactivate( void );
[id(0x0000000A)]
void OnKeyPress([in, out] short * Key );
[id(0x0000000F)]
void OnPaint( void );
};

[
  uuid(A70D6152-C9F6-11D5-91DF-0000E85EFBBA),
  version(1.0),
  helpstring("ActiveFormX Control"),
  control
]
coclass ActiveFormX
{
  [default] interface IActiveFormX;
  [default, source] dispinterface IActiveFormXEvents;
};

[
  uuid(A70D6154-C9F6-11D5-91DF-0000E85EFBBA),
  version(1.0)
]
typedef enum tagTxActiveFormBorderStyle
{
  [helpstring("afbNone")]
  afbNone = 0,
  [helpstring("afbSingle")]
  afbSingle = 1,
  [helpstring("afbSunken")]
  afbSunken = 2,
  [helpstring("afbRaised")]
  afbRaised = 3
} TxActiveFormBorderStyle;

```



```
[  
  uuid(A70D6155-C9F6-11D5-91DF-0000E85EFBBA),  
  version(1.0)  
]
```

```
typedef enum tagTxPrintScale
```

```
{  
  [helpstring("poNone")]  
  poNone = 0,  
  [helpstring("poProportional")]  
  poProportional = 1,  
  [helpstring("poPrintToFit")]  
  poPrintToFit = 2  
} TxPrintScale;
```

```
[  
  uuid(A70D6156-C9F6-11D5-91DF-0000E85EFBBA),  
  version(1.0)  
]
```

```
typedef enum tagTxMouseButton
```

```
{  
  [helpstring("mbLeft")]  
  mbLeft = 0,  
  [helpstring("mbRight")]  
  mbRight = 1,  
  [helpstring("mbMiddle")]  
  mbMiddle = 2  
} TxMouseButton;
```

```
[  
  uuid(A70D6157-C9F6-11D5-91DF-0000E85EFBBA),  
  version(1.0)  
]
```

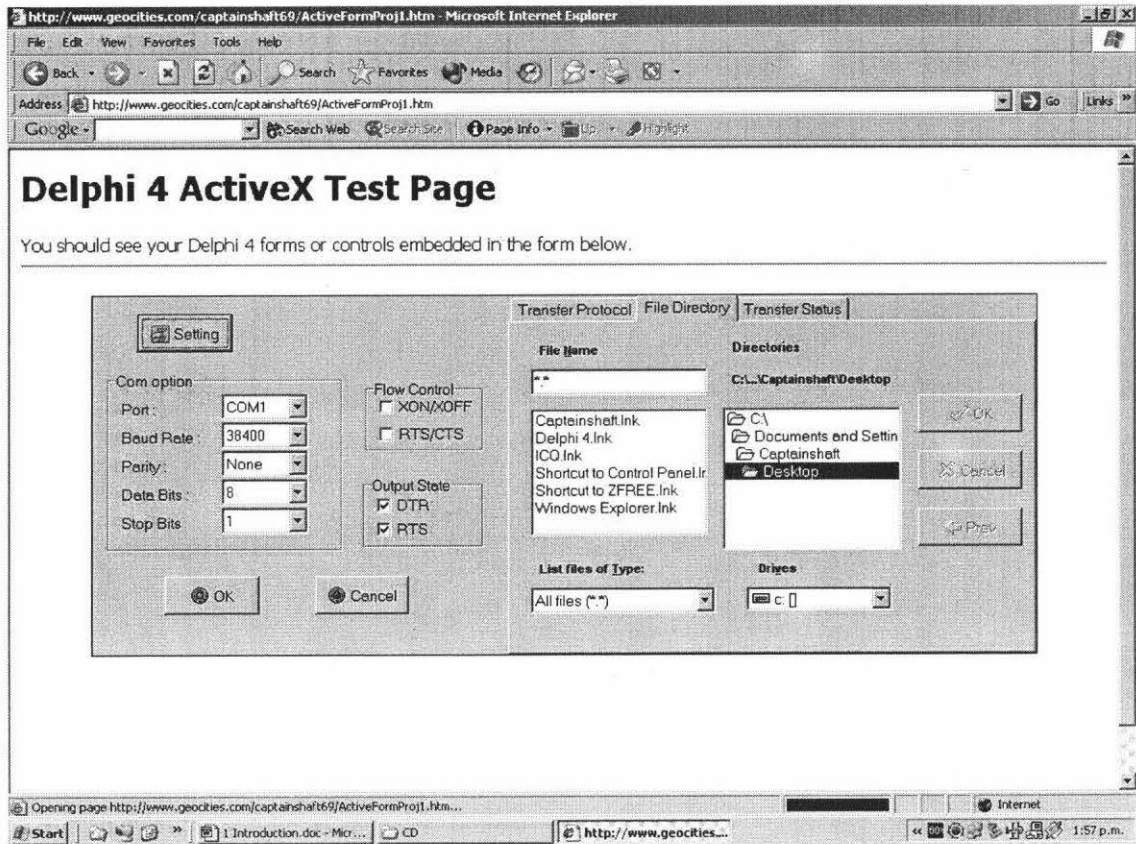
```
typedef enum tagTxBiDiMode
```

```
{  
  [helpstring("bdLeftToRight")]  
  bdLeftToRight = 0,  
  [helpstring("bdRightToLeft")]  
  bdRightToLeft = 1,  
}
```

```
[helpstring("bdRightToLeftNoAlign")]
bdRightToLeftNoAlign = 2,
[helpstring("bdRightToLeftReadingOnly")]
bdRightToLeftReadingOnly = 3
} TxBiDiMode;

};
```

17. ACTIVEPROJ1.HTML



```
<HTML>
```

```
<H1> Delphi 4 ActiveX Test Page </H1><p>
```

```
You should see your Delphi 4 forms or controls embedded in the form below.
```

```
<HR><center><P>
```

```
<OBJECT
```

```
  classid="clsid:A70D6152-C9F6-11D5-91DF-0000E85EFBBA"
```

```
  codebase="C:/CD/ActiveFormProj1.dll"#version=1,0,29,0
```

```
  width=847
```

```
  height=321
```

```
  align=center
```

```
  hspace=0
```

```
  vspace=0
```

```
>
```

```
</OBJECT>
```

```
</center></HTML>
```