ELECTRO STATIC PRECIPITATOR
NTEGRATED OPERATING SYSTEM

Project Report 2001 – 2002

P-727

Project work done by

BHARATHIRAJA.S,

Under The Supervision and Guidance of
Mr P.Gopalakrishnan MCA.,
Lecturer,
epartment of Computer Science and Engineering)

In partial fulfillment of the requirements for the Award of degree of Master Of Science-Applied Sciences (Computer Technology) of the Bharathiar University, Coimbatore-46 during the year 2001-2002.

PARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECNOLOGY
COIMBATORE 06

Coimbatore 641 006 LIBRARY KCT

CERTIFICATE

BHARAT HEAVY ELECTRICALS LIMITED

(A Govt. of India Undertaking)
Boiler Auxiliaries plant
Ranipet 632406

CERTIFICATE

Certified that Shri **Bharathiraja.s MSc [Applied Science Computer technology]** student of Kumaraguru College of Technology under Bharathiar university have done a project on the title **"Electro Static Precipitator Integrated Operating system in VC++"** during 25-12-2001 to 31-3-2002 in BHEL, Ranipet in Partial fulfillment of his study.

R.Lakshmanan.
Manager/AQCS
(Guide)
R. LAKSHMANAN,
Manager,
Air Quality Control Systems
Engg. & Development Centre
BHEL / BAP / RANIPET - 632 406

S.Paramananthan.
SPO/IR&HRC
S. PARAMANANTHAN,
Sr. Personnel Officer / IR & HRD
BHEL/BAP/RANIPET - 632 406.

DECLARATION

DECLARATION

I hereby declare that this project entitled

ELECTRO STATIC PRECIPITATOR INTEGRATED OPERATING SYSTEM

Is submitted in partial fulfillment of the requirement for the award of the degree of M.Sc [Applied Science – Computer Technology] is the report of the original work done by me during the period of study (2001-2002) in

KUMARAGURU COLLEGE OF TECHNOLOGY
CHINNAVEDAMPATTI
COIMBATORE

Under the supervision of

Mr. P. Gopalakrishnan. M.C.A.,
Lecturer,
(Department of Computer Science And Engineering)

Name	Register Number	Signature
Bharathiraja.S	**0037Q0030**	Bharathiraja

Place : **Coimbatore**
Date : 24-04-2002.

ACKNOWLEDGEMENT

CONTENT

CONTENTS

INTRODUCTION

1.1 ORGANISATION PROFILE

BHEL -An Overview

BHEL is the largest engineering and manufacturing enterprise in India in the energy related/infrastructure sector today. BHEL was established more than 40 years ago when its first plant was set up in Bhopal ushering in the indigenous Heavy Electrical Equipment Industry in India, a dream which has been earning profits continuously since 1971-72 and achieved a sales turnover of Rs.6347 crore with a pre-tax profit of Rs.294 crore in 2000-2001.

Power Generation

Power Generation Sector comprises thermal, gas, hydro and nuclear power plant business. The company manufactures 235 MW nuclear turbine-generator sets, and has commenced production of 500 MW nuclear turbine generator sets. Custom-made hydro sets of Francis; Pelton and kaplan types for different head-discharge combinations are also engineered and manufactured by BHEL.

Power Transmission & distribution (T&D)

BHEL offers wide-ranging products and systems for T&D applications. The products, which are manufactured, include: Power transformers, instrument transformers, dry type transformers, series and shunt-reactors, capacitor banks.

Industries

BHEL is a major contributor of equipment and systems to industries like cement, sugar, fertilizer, refineries, Petrochemicals, paper, oil and gas, metallurgical and other process industries. The range of systems & equipment supplied includes: captive power plants, co-generation plants, DG power plants, industrial steam turbines, industrial boiler and auxiliaries, waste heat recovery boilers, gas turbines, heat exchanges and pressure vessels, centrifugal compressors, electrical machines, pumps, valves, seamless steel tubes, electrostatic precipitators, fabric filters, reactors, fluidized bed combustion, chemical recovery boilers and process controls.

Transportation

BHEL is involved in the development design, engineering, marketing, production, installation, and maintenance and after-sales service of rolling stock and traction propulsion systems. In the area of rolling stock, BHEL manufactures electric locomotives up to 5000 HP, diesel-electric locomotives from 350 HP to 3100 HP, both for mainline and shunting duty applications. BHEL is also producing rolling stock for special applications vice., overhead equipment cars, Special well wagons, Rail-cum-road vehicle etc. Beside traction propulsion system for in-house use, BHEL manufactures traction propulsion systems for other rolling stock producers of electric locomotives, diesel – electric locomotives, electrical multiple units and metro cars.

Environmental Policy

- Compliance with applicable Environmental Legislation/Regulation;
- Continual improvement in Environment Management Systems to protect our natural environment and control pollution;
- Promotion of activities for conservation of resources by Environmental Management;

1.2 SYNOPSYS

The project titled "ESP-IOS (ElectroStatic Precipitator-Integrated Operating System) is a Windows based system mainly being Designed for controlling and centrally monitor status of the devices BAPCON and RAPCON through a PC.

The ESP system operates with real time data displaying the current status of the BAPCONs and RAPCONs With respect to the input values. Electrostatic Precipitator Integrated Operating System (ESP-IOS) is a PC based Management system developed for ESP applications and is one Of the most sophisticated system. Controlling an ESP with IOS Ensures an effective control of the entire process in the ESP. Remote control of entire ESP operation can be achieved from a Single point. The set point of the various BAPCONs, RAPCONs Can be changed from IOS. The status of the ESP and print out of The same can be obtained in the IOS PC.

- Maximum dust collection, by optimizing charge ratio.
- Minimum power consumption with allowed emission, by Increasing charge ratio and reducing current.
- Automate ESP startup and shutdown procedure at Minimum time and cost.

The IOS is adaptable to the varying requirements of the Different objectives like above. Incorporates specialized sub Controllers, which are independent and can control and Supervise their part following their local parameters. The Comprehensive control of the precipitator can be set in the IOS.

On line help is available in the IOS with easy access.

Entire software system of the IOS (Integrated Operating System) is programmed for running in Microsoft Windows NT 4.0 or later operating system. The communication between IOS and sub controllers, BAPCON and RAPCON is made through two serial communication RS 232C ports in this port1 of the serial communication is used for communication with DATA LOGGER PC which is an optional choice.

1.3 PROBLEM DEFINITION

In order to succeed and to meet the expectations of management, every application-development project should begin with an analysis of the requirements and a clear list of goals.

A system is an orderly grouping of interdependent components linked together according to a plan to achieve a specific objective. Analysis may be defined as the process of dividing into parts, identifying each part and establishing relationship with in the parts. It is the process of studying a problem to find the best solution to that problem. The problem that exists in the system is least and the requirements are defined and the solution is evaluated.

This project "Electro Static Precipitator Integrated Operating System" should be for setting the parameters of the sub controllers BAPCON and RAPCON and also to display the status of the sub controllers, which is updated as per the timer setting for each of the sub controllers. There should be good security measures for the

Project. There should be two levels of password settings one for the user level and other for the official level.

Project Function definition

- Should be capable of using the Microprocessor technology component hardware and software.
- Functions for precipitator operation.
- Ability to tailor the electrical operation and rectify the actual real-time precipitator operation condition.
- Fast response
- Separate control Strategy

SYSTEM ANALYSIS

2.SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

The existing system is manual. Problem is time consuming as well as less security. All the information is maintained manually. Manual operation takes more time. The Existing system is a DOS based system and is not user friendly. It did not have proper security system. It did not have proper display system with this the existing system did not have the following features too

- It should be capable of using the Microprocessor technology component hardware and software.
- Functions for precipitator operation.
- Ability to tailor the electrical operation and rectify the actual real-time precipitator operation condition.
- Fast response
- Separate control Strategy

2.2 Proposed system

The proposed system is a computerized one. Response time of the new system is very quick compared to the old system. Management information report on ESP made online. This program centrally monitors the status of ESP BAPCON and RAPCON controller on demand for the user in real time. Some of its improved features are,

Rs 232C Serial Communication.

User Friendly GUI features.

Compatible with various printers.

On line help features.

Less storage space.

Reliable and flexible.

Since it is developed in a object oriented programming language it is reusable and extendable.

P 727

LIBRARY
641 006

Security envisaged in the system will ensure that only the authorized people do usage of the system. It does have a quick response to the input. A good alarm system is enforced to tell if any failure of the working system in the sight. (For e.g. failure of rapper motors). If the system is repaired it also can be reset for further use. Entire software system of the IOS (Integrated Operating System) is programmed for running in Microsoft Windows NT 4.0 or later Operating System. The communication between IOS and sub controllers BAPCON and RAPCON is made through two serial communication RS 232C ports in this port1 of the serial communication is used for communication with DATA LOGGER PC which is an optional choice.

GM of the power station can monitor the situation easily without moving anywhere. Different IOS status can be viewed.

The new system has the following facilities.

- It is capable of using the Microprocessor technology component hardware and software.
- Functions for precipitator operation.
- Ability to tailor the electrical operation and rectify the actual real-time precipitator operation condition.
- Fast response and Separate control Strategy

DEVELOPMENT ENVIRONMENT

3. DEVELOPMENT ENVIRONMENT

3.1 SOFTWARE REQUIREMENTS

- Microsoft Visual C++ 6.0.
- Windows NT 4.0 or Above.
- ISA Card Port Reading and NT Port I/O Device Driver installation.
- Graphics server 5.0 for Windows for graphing.
- Help Compiler Such as

  1 ROBO win help compiler.

  2 HTML Help Workshops.

3.2 SOFTWARE SPECIFICATION

MICROSOFT VISUAL C++

Microsoft Visual C++ consist of two parts

1. Software development Kit (SDK).
2. Microsoft Foundation classes (MFC).

1.Software Development

It is a part of Visual C++, which helps in developing programming any sort of windows based programming can be done using SDK such as

- Windows
- Buttons
- Dialogs
- List Boxes, combo boxes etc
- Controls, bitmaps, cursors, string menus, accelerators.

Such things could be created using the Software Development Kit (SDK); with this functions for these created items could also be written in SDK.

. Microsoft Foundation Class

story and Evolution

e Microsoft Foundation Class Libraries a robust C++ application framework designed writing applications in C++ for the Microsoft Windows operating system. Is the latest the growing and evolving library.MFC was released in April 1992 with Microsoft C++ 7. The 32-bit was released later that year as part of the Win32® Prerelease velopment Kit program. The features of MFC fell mainly into two categories: general-rpose classes for the nongraphical portion of an application and Windows-related sses for the graphical user-interface (GUI) features of an application.
eneral-purpose classes were the following:

- Run-time type information
- Object persistence
- Collection classes
- Strings
- Files
- Time and date
- Exception handling
- Exception handling

ndows-related classes were the following:

- Application startup and other application services
- Window management
- Graphics device interface (GDI)
- Multiple-document interface (MDI)
- Menus
- Dialog boxes
- Windows controls
- Windows common dialogs
- OLE

hitectural classes were the following:

- Commands
- Documents and views
- Printing and print preview
- Dialog data exchange and validation (DDX/DDV)
- Context-sensitive help

gh-level abstractions were the following:

- Form view
- Edit view
- Scrolling view
- Splitter window
- Toolbars and status bar
- Dialog bar and other control bars
- VBX controls (16-bit only)

tabase classes were the following:

- Database engine classes
- Record field exchange (RFX)
- Record view

LE classes were the following:

- Visual editing servers
- Visual editing containers
- Drag and drop
- Structured storage
- OLE Automation servers
- OLE Automation clients

ew user-interface classes were the following:

- Enhanced toolbars
- Mainframe windows
- Tabbed dialogs (property pages)

New support for Win32 consisted of the following:

- New Win32 APIs
- Multithreading
- Unicode™ support
- Shared 32-bit dynamic-link libraries (DLLs)

New language syntax support included the following features:

- C++ templates
- C++ exceptions
- Windows 95 common controls
- Simple MAPI
- Windows Sockets
- Swap-tuned DLL s
- Containment of OLE controls
- Data Access Objects (DAO)
- Simplified Windows 95 common controls
- Windows 95 common dialogs
- Thread synchronization objects

Features

Portability

Portability has been designed to be portable to a number of platforms, allowing applications written to target a variety of different platforms. Is also portable to a variety of compiler implementations, and the many companies who have licensed have made it a standard application framework for Windows-based development. Microsoft plans to keep extending and evolving to support new functionality for applications and to exploit new functionality in the operating system.

Multiple Platforms

Writing a program on top of makes that program portable to a wide variety of platforms. Because is built on top of the Win32 API, any platform that supports the Win32 API can be targeted by an application. A powerful aspect of portability is that Win32-based platforms are not limited to Microsoft Windows. Through the use of the Windows Portability Layer, applications built on Win32 can also be built to run under the Macintosh System 7 operating system.

applications can target a large number of platforms today:

- Microsoft Windows 95
- Microsoft Windows NT running on Intel® processors
- Microsoft Windows NT running on MIPS® processors
- Microsoft Windows NT running on Alpha AXP™
- Microsoft Windows NT running on PowerPC™
- Microsoft Windows 3.1 using the Win32s® API
- Apple® Macintosh (using the 68000-series instruction set)
- Apple Macintosh (using the PowerPC™ instruction set)

Windows Common Control Classes

Microsoft Visual C++ classes to encapsulate most of the new Windows common controls supplied with the Windows 95 and Windows NT operating systems. The integrated tools (editors and wizards) in Visual C++ also support Windows common controls. These controls are also supported by the Visual C++ Cross-Development system for the Macintosh.

The Windows common controls supported by including the following:

- **CAnimateCtrl.** A control that displays successive frames of an Audio Video Interleaved (AVI) clip during a lengthy operation.
- **CCheckListBox.** A control that displays a list of items, such as file names, that the user can view and select. A check box appears next to each item in the list; the user can check or clear the selected item's check box.
- **CDragListBox.** A control similar to a **CListBox** that allows the user to move items, such as file names and string literals, within the list box. List boxes with this capability are useful for an item list that is in an order other than alphabetic, such as one that includes path names or files in a project.

- **CHeaderCtrl**. A resizable button that appears above a column of text, allowing the user to display more or less information in the column.

- **CHotkeyCtrl.** A window that enables the user to create a hot key. A *hot key* is a key combination that the user can press to perform an action quickly.

- **CImageList.** A collection of images used to efficiently manage large sets of icons or bitmaps.

- **CListCtrl.** A window that displays a collection of items, each consisting of an icon and a label.

- **CProgressCtrl.** Also known as a *progress bar control*, this window can be used by an application to indicate the progress of a lengthy operation.

- **CRichEditCtrl.** A window in which the user can enter and edit text with character and paragraph formatting. The control can include embedded OLE objects.

- **CSliderCtrl.** Also known as a *trackbar*, this window containing a slider and optional tick marks sends notification messages to indicate changes in its position.

- **CSpinButtonCtrl.** Also known as an *up-down control*, this pair of arrow buttons can be clicked to increment or decrement a value, such as a scroll position or a number displayed in a companion control.

- **CStatusBarCtrl.** A horizontal window in a parent window in which an application can display various kinds of status information. This control resembles the **CStatusBar** class.

- **CTabCtrl.** This control is analogous to the dividers in a notebook or the labels in a file cabinet. By using a tab control, an application can define multiple pages for the same area of a window or dialog box.

- **CToolBarCtrl.** A window that contains one or more command-generating buttons. This control resembles the **CToolBar** class.

- **CToolTipCtrl.** A small pop-up window that displays a single line of text describing the purpose of a toolbar button or other tool in an application.

- **CTreeCtrl.** Also known as a *tree view control*, this window displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional bitmapped image, and each item can have a list of subitems associated with it.

veral **CView**-derived classes are also available to simplify the use of the new controls document-view architecture applications. **CListView**, **CTreeView** and

RichEditView make it easier to use the **CListCtrl**, **CTreeCtrl** and **CRichEditCtrl** jects. To provide further support for rich-text editing, **CRichEditDoc**, **CRichEditView** d **CRichEditCntrItem** classes are available that provides Document/View and control capsulation.

ull OLE Control Support

Now supplies complete OLE control container support and integrates the OLE ontrol Development Kit (CDK) with the rest of . This allows developers to build, e, and share OLE controls with other developers, even those using different tools. evelopers can choose from hundreds of OLE controls already on the market or build eir own. These OLE controls are compatible with the latest s of the Microsoft Visual asic® programming system and the Microsoft Visual FoxPro™ database management stem, as well as with many other development tools. The Visual C++ dialog editor upports placing OLE controls in a dialog resource, making it easy to use prebuilt ontrols in applications. With the new OLE controls container, developers need not nderstand all the details of using OLE container applications. Support is based on the **Wnd** class, which allows users to create both the container and the control sides. In n OLE control becomes a special kind of child window, with **CWnd** functions, including **Wnd::CreateControl**, which dynamically creates an OLE control rather than an rdinary window. Support is also provided for dialog data exchange (DDX), preloaded ontrols for improved performance, and transparent keyboard translation in **DialogMessage**.

Direct Database Access with DAO

In addition to the existing ODBC database support, the new Data Access Object DAO) classes for enable developers to directly access the Microsoft Jet database ngine, which is the same engine that is used in Microsoft Access for Windows 95 and Microsoft Visual Basic 4.0. The DAO classes encapsulate an OLE Component Object Model (COM) interface to the Jet database engine, so developers don't have to write the tructured query language (SQL) themselves unless they want to do so.
As with the ODBC database classes, developers can open and manipulate databases nd recordsets and display the data in a form view. With DAO, they can also set up and se work spaces, create and manipulate tables and queries, and use and create ndexes for those tables through the use of the SQL data definition language (DDL). Developers can also access ODBC data sources through attached tables using the DAO classes and the Jet engine.

New Common Dialogs

Two new OLE common dialog classes, **CPageSetupDialog** and **COlePropertiesDialog**, have been developed to improve access to several aspects of an OLE object. **COlePropertiesDialog** encapsulates the Windows common OLE Properties dialog box to provide an easy way to display and modify the properties of an OLE document item in a manner consistent with Windows specifications.

In addition, **CPageSetupDialog** encapsulates the services provided by the Windows common Page Setup dialog box with additional support for setting and modifying print margins. Several other new common dialogs, including the Windows 95 file I/O dialogs, are now supported by.

Thread Synchronization Objects

Multithread programs often require synchronizing access to shared resources by different concurrent threads. To manage synchronization, supplies a new base class, **CSyncObject**, and several derived objects that represent common synchronization techniques. **CSyncObject** provides functionality common to the derived synchronization objects in the Win32 API. Support includes **Lock** and **Unlock** as virtual abstract operations that derived classes override. Classes derived from **CSyncObject** are **CSemaphore**, **CCriticalSection**, **CMutex**, and **CEvent**, encapsulating the Win32 synchronization objects. The **CSingleLock** and **CMultiLock** classes represent the access-control mechanisms used in controlling access to a resource in a multithread program.

Simple MAPI Support

MAPI, the Messaging API for Windows, is a set of functions that mail-enabled and mail-aware applications use to create, manipulate, transfer, and store mail messages. It gives application developers the tools to define the purpose and content of mail messages, and it gives them flexibility in their management of stored mail messages. MAPI also provides a common interface that application developers can use to create mail-enabled and mail-aware applications independent of the underlying messaging system.

Messaging clients provide a human interface for interaction with MAPI. This interaction typically includes requesting services from MAPI-compliant providers such as message stores and address books.

3.1 includes support for Simple MAPI in classes **CDocument** and **COleDocument**. This support is also provided in 2 for 16-bit development. With minimal effort, developers can add the ability to send an application's documents by means of the resident e-mail host. (Sending OLE compound documents is managed correctly in the **COleDocument** portion of the MAPI implementation.)

Support for Windows Sockets

New Windows Sockets classes have been added for network programming. provides support for Windows Sockets, the network-independent API for network communications programming under the Microsoft Windows and Windows NT operating systems.The new classes include **CAsyncSocket**, **CSocket**, and **CSocketFile**. Class **CAsyncSocket** encapsulates the Windows Sockets API. Class **CSocket**, derived from **CAsyncSocket**, additionally provides a simple programming model that lets developers

serialize data from one socket application to another via a **CArchive** object, using a **CSocketFile** object.

Enhanced Toolbars

One of the most commonly requested user-interface elements is the toolbar, a row of buttons represented by bitmaps and optional separators. These bitmap buttons can behave like push buttons, check-box buttons, or radio group buttons. The class **CToolBar** supports the standard toolbar look. All the toolbar buttons are normally taken from a single bitmap image, which is edited using the Visual C++ bitmap editor and contains one image for each button. Storing all the images in one bitmap reduces the amount of system resources used by an application.

One of the key advantages of the MFC **CToolBar** class is that by using commands, programmers can enable and disable the various buttons in the toolbar in conjunction with any menu items for those same commands. This is important because toolbar buttons almost always-duplicate menu items, allowing the programmers to write the command handler once and drive it from either a menu item or a toolbar button.

adds dynamic resizing and layout of toolbars, which lets users change not only the oolbar location, but also resize the toolbar window. This makes vertical toolbars and ool "palettes" possible. A dockable toolbar can be attached or "docked" to any side of ts parent window, or it can be "floated" in its own miniframe window (using **CMiniFrameWnd**).

Programmers who use AppWizard to generate the skeleton of an application are asked o choose whether or not they want dockable and resizable toolbars. By default, AppWizard creates code to enable docking, resizing toolbars. **CToolBar** and **CFrameWnd** member functions are available to customize the behavior of the docking oolbar and to programmatically dock or float a toolbar.

Support for "tool tips" is also included. When the user moves the mouse over a toolbar utton, a small box is shown on top of the button to describe the action that would be erformed. Also supported are "fly-by" tool tips that provide a more detailed description f the command on the status bar. This saves the user from having to press a toolbar utton to find out what the command does. MFC also allowed persistence of toolbar onfigurations. Like many professional applications, MFC allows users to save the ntire state of a given frame window's toolbar configuration, including each toolbar's urrent position and visible and floating states.

he **CToolBar** class can easily support additional standard Windows controls, such as op-down list boxes or edit controls, on the toolbar. In addition, **CToolBar** provides rogrammatic APIs for dynamically changing the buttons on the toolbar, customizing ocking behavior, and highly customizing user interfaces in other ways.

iniframe Windows

Miniframe windows are frame windows with thin caption bars, like those used in e Visual C++ property windows.The MFC class **CMiniFrameWnd**, derived from **FrameWnd**, provides an alternative user interface for floating palettes and toolbars. In ct, the **CToolBar** implementation of tear-off toolbars uses the **CMiniFrameWnd** class

hold the torn-off toolbar.A **CMiniFrameWnd** object represents a half-height frame ndow typically seen around floating toolbars. These miniframe windows behave like rmal frame windows, except that they do not have minimize/maximize buttons or enus, and the user only has to single-click the system menu to dismiss them.

roperty Sheets

Contains support for *property sheets*, also known as "tabbed dialog boxes." A operty sheet is a special kind of dialog box that is generally used to modify the tributes of some external object, such as the current selection in a view. The property eet has three main parts: the containing dialog box, one or more property pages own one at a time, and a tab at the top of each page that the user clicks to select that ge. Property sheets are useful when a number of similar groups of settings or options ed to be changed. An example of a property sheet is the Project Settings dialog box Visual C++. In this case, a number of different groups of options need to be set. The operty sheet allows a large amount of information to be grouped in an easily derstood fashion.This support is provided in two classes: **CPropertySheet**, which is a ass to contain all the pages, with one tab per page; and **CPropertyPage**, which is a ass that each property page is derived from.To create a property sheet with several ges, first create a dialog template resource for each property page using the Visual ++ dialog editor, then use ClassWizard to create a **CPropertyPage**-derived class rresponding to each property page dialog template. For each of these new classes, se ClassWizard to create member variables to hold the values for the property page. he process for adding member variables to a property page is exactly the same as lding member variables to a dialog box, because a property page is a specialized alog box. Creating the property sheet at run time is easy to do either by using the **PropertySheet** class directly or by deriving a more specialized property sheet from it. or Windows 95, Is designed so that applications run with the correct user interface epending on which operating system they are running on.Developers can build one in32-based executable file using. The application will look and act like a Windows 3.1- sed application when run on Windows 3.1 or on Windows NT on an Intel processor. hen the same executable file runs under Windows 95, the application will look and act e a Windows 95-based application. Splitter windows, toolbars, status bars, and iniframe windows all have a markedly different look when running on Windows 95. so, **CScrollView**-derived classes automatically take advantage of proportional scroll umbs, if available. In addition, the property page classes were modeled after the indows 95 user interface.

upport for Win32

Continues to play its role as the C++ API to Windows. MFC provided expanded nd improved encapsulation of some of the Win32 services. Member functions in isting classes improved the Win32 API coverage. Classes support writing multithread pplications, Unicode, or Double-Byte Character Sets (DBCS) applications using a shared DLL.MFC also included extended coverage of the Win32 API, including DI functionality such as Béziers, Paths, and a number of other Win32 USER APIs.

Multithreading

Is thread-safe and supports writing multithread applications. Threads of execution are encapsulated in the class **CWinThread**. The main application class, **CWinApp**, is derived from **CWinThread**; it represents the main user-interface thread of the application.

distinguishes two types of threads: user-interface threads and worker threads. A user-interface thread is commonly used to handle user input and respond to events and messages generated by the user. Worker threads are commonly used to complete tasks that do not require user input, such as recalculation. Handles user-interface threads specially by supplying a message pump for events in the user interface. **CWinApp** is an example of a user-interface thread object, because it derives from **CWinThread** and handles events and messages generated by the user.
Developers can create additional threads in their applications if they wish, creating new objects of the class **CWinThread** or a class derived from **CWinThread**. In most situations, the developer doesn't even have to create these objects explicitly and can instead call the framework helper function **AfxBeginThread**, which creates the **CWinThread** object.
Of course, even with the multithread enabling of , writing and debugging multithread applications is an inherently complicated and tricky undertaking, because the developer must ensure that a given object is not accessed by more than one thread at a time.

Unicode Support

Some international markets use languages that have large character sets, such as Japanese and Chinese. To support programming for these markets, the library is enabled for two approaches to handling large character sets, Unicode and Double-Byte Character Sets (DBCS).
DBCS is supported on all platforms. Unicode is supported on all Windows NT platforms. The library is designed to support either option. A special of the library must be linked in when building a Unicode application. MFC is provided in a directly supports Unicode characters and strings. In particular, class **CString** is Unicode-enabled. **CString** is based on the **TCHAR** data type. If the symbol _UNICODE is defined for a build of the program, **TCHAR** is defined as type **wchar_t**, a 16-bit character-encoding type; otherwise, it is defined as type **char**, the normal 8-bit character encoding. As a result, under Unicode, **CStrings** are composed of 16-bit characters; without Unicode, they are composed of characters of type **char**. **CString** also supplies Unicode-aware constructors, assignment operators, and comparison operators.
If Unicode is not specified, the class library defaults to supporting the ANSI character set, with support specifically for DBCS. Under the DBCS scheme, a character can be either one or two bytes wide. If it is two bytes wide, its first byte is a special "lead" byte, chosen from a particular range depending on which code page is in use. Taken together, the lead and "trail" bytes specify a unique character encoding.

n either case, CString con operators and constructors make it easy to convert ANSI nd Unicode strings to a **CString** object, in the case that program deals with both ANSI nd Unicode characters.Note that **Unicode** string serialization in can read both Unicode nd DBCS strings, regardless of which of the application is running. Because of this, ata files are portable between Unicode and DBCS s of the program.

Shared 32-Bit DLLs

provides the entire application framework in a shared DLL form. Introduced the hared DLL feature and provided a single DLL for all the features (MFC200.DLL). This eature was continued with (250.DLL). However, this shared DLL was available only on ne 16-bit Windows platform.
The functionality was factored into three 32-bit DLLs: one for the core features MFC30.DLL), one for the OLE 2-specific support (MFCO30.DLL), and one for the latabase-specific support (MFCD30.DLL). All three DLLs are available in two forms: a reely redistributable retail and a debugging.
Combines all of these DLLs into one unified DLL for simplified distribution. For ncreased performance, advanced page-tuning techniques reduce overhead for pplications that don't use all features.
The benefits of this shared DLL implementation to the programmer are significant. Now, several applications can share framework code if they are running simultaneously, thus educing system-resource usage. Also, executable files are much smaller than static-ink s of the same applications, although the size of the DLL must be included for a true neasurement.

Language Syntax Support

takes advantage of new C++ language features such as C++ templates and exceptions. MFC was designed at the time these C++ language features were still under development, so additional tools and macros were needed. Today, with and direct compiler support for these language features, the benefits of the original design are visible.With there is no need to use **TEMPLDEF** or the **TRY**, **CATCH**, and **THROW** exception macros. True C++ syntax for templates and "templatized" collections are supported. C++ exceptions are supported for both new and old code, so objects on the stack will always be destroyed when exceptions are thrown. For backward compatibility, he old ways of doing things are still supported.

C++ templates

provides collection classes based on C++ templates, which makes it easier to derive one's own type-safe collection classes. MFC provided two types of collection classes to manage groups of objects: collection classes that are created from C++ emplates; and Collection classes that are not created from templates.
The contemplate collection classes are the same as those provided by MFC. Developers whose code already uses these classes can continue to use them.

velopers who write new type-safe collection classes for their own data types should nsider using the newer template-based classes.
collection class is characterized by its *shape* and by the types of its elements. The ape refers to the way the objects are organized and stored by the collection.
ovides three basic collection shapes: lists, arrays, and maps (also known as tionaries), which provide easy-to-understand yet powerful building blocks for the rest the application. Developers can pick the collection shape most suited to their rticular programming problems:

hape	Description
st	The list class provides an ordered, nonindexed list of elements, implemented as a doubly linked list. A list has a "head" and a "tail," and adding or removing elements from the head or tail, or inserting or deleting elements in the middle, is very fast.
rray	The array class provides a dynamically sized, ordered, and integer-indexed array of objects.
ap	A map is a collection that associates a key object with a value object.

e easiest way to implement a type-safe collection that contains objects of any type is use one of the template-based classes.

ollection contents	Arrays	Lists	Maps
ollections of objects of any type	CArray	CList	CMap
ollections of pointers to objects f any type	CTypedPtrArray	CTypedPtrList	CTypedPtrMap

r example, to create a dynamically sized array of "doubles," use C++ template ntax:

```
Array<double, double> myArray;
```

he application already uses nontemplate classes, developers can continue to e them, but they should consider using the template-based classes for new llections.

++ exceptions

Exceptions are used to signal abnormal execution, including situations in which nditions outside the program's control are influencing the outcome of the function, ch as low memory or I/O errors. Abnormal situations are handled by catching and rowing exceptions, rather than by using return codes that are often overlooked and so sult in inefficient code. The exception syntax is a clean and efficient mechanism for normal conditions. MFC provides two compatible ways of using exceptions: the C++ ceptions, available starting with and the exception macros, available starting with evelopers writing new applications using should use the C++ exception mechanism. he macro-based mechanism can be used if the existing application already uses that echanism extensively. Existing codes can be converted readily to use C++ exceptions stead of the exception macros, if desired. If an application has already been eveloped using the exception macros, the developer can continue to use the exception acros in the existing code, while using C++ exceptions in new code.

Whether they use the C++ exceptions directly or use the exception macros, developers will use **CException** or **CException**-derived objects that may be thrown by the framework or by the application. MFC provides several predefined exception classes to handle everything from out-of-memory to OLE dispatch exceptions.

Database Classes

Visual C++ provides integrated support in AppWizard, ClassWizard, and the Visual C++ dialog editor for building database applications. In, provided database engine functionality. Also, a new mechanism was provided to enable data-bound controls using ClassWizard, which makes it easy for the developer to build forms that view and edit fields in a database without having to write any C++ code.
These database classes provide easy access to any database for which an ODBC driver is available. Also provided with Visual C++ are portions of the ODBC SDK, the ODBC driver manager, and a new set of drivers.

Database engine classes

In addition to the new DAO classes, there are two main classes that provide the encapsulation of ODBC data sources: **CDatabase** and **CRecordset**. These classes are modeled after the high-level database abstractions used in the Visual Basic programming system and the Microsoft Access database management system. The of hese abstractions are provided as type-safe C++ classes that allow programmers to ccess data in an easy and comfortable way, without requiring them to give up the erformance and compile-time checking that C++ provides.
he class **CDatabase** represents a connection to a data source through which evelopers can operate on the data source. The database engine classes are smart nough to allocate and open a new **CDatabase** connection as needed by the program. evelopers who want to manage their database connections manually, or to optimize erformance or control transactions, will want to use the **CDatabase** class.
he class **CRecordset** encapsulates a set of records selected from a data source. ecordsets enable scrolling from record to record, updating of records (adding, editing, d deleting records), qualifying the selection with a filter, sorting the selection, and arameterizing" the selection with information obtained or calculated at run time. ogrammers will normally derive their own custom classes from **CRecordset** to add pe-safe member variables to the class that will be bound to specific columns of a data urce. The process of creating a new C++ class and managing the binding of columns member variables is performed by ClassWizard. The process of moving data tween the database and the member variables of the class is called *record field change* (RFX). This technique is similar to the dialog data exchange (DDX) echanism introduced in Data values obtained from an ODBC data source are turned o C++ and data types such as short, long, float and **CString**. For large binary data, a ss **CLongBinary** is provided for efficient management of large data values.

Record view

One of the most common types of Windows-based applications is form processing. A form is like a dialog box that the user can interact with to fill in edit controls, select options from list boxes and radio groups, and work with other dialog box controls. For example, an order/entry database application would probably use forms to allow customer service representatives to enter order information.
introduced the high-level abstraction of a form view. MFC included a class called **CRecordView**, derived from **CFormView**, that provides a form directly connected to a recordset object.
Using the same mechanism as for dialog boxes and simple form views, a record view uses the DDX mechanism to exchange data values between the recordset and the controls of the record view. Like all form views, a record view is based on a dialog template resource. Record views also support moving from record to record in the recordset, updating records, and closing the associated recordset when the record view closes.
Like any form view, a record view automatically supports many features, such as scrolling, synchronous update, support for VBX custom controls, support for standard Windows controls, and the ability to place a form view in any window, including MDI child windows.

OLE Classes

OLE provides many attractive features to the end user of OLE-capable applications. For the developer, OLE provides a lot of functionality in many APIs but adds the burden of many design decisions and a lot of implementation work.The support for OLE provides C++ classes integrated with the framework that make developing OLE-capable applications easy. AppWizard has been extended to make it easy to start developing OLE applications. ClassWizard has been extended to make it easier to support OLE Automation in programs.The support for OLE encapsulates much of the complexity of the OLE API in a small set of C++ classes that provide a higher-level interface to OLE. Of course, following the design philosophy, developers can call the underlying C-language OLE API functions directly wherever the OLE classes don't meet their needs.Also provided with Visual C++ is the OLE SDK, as well as a number of tools and sample applications to help developers test their OLE applications. Online documentation includes overview material, tutorials, an encyclopedia, and a class reference.This paper divides the features of OLE into two main categories: features for supporting OLE visual editing and features for OLE Automation. The development work involved depends on whether the application will provide these services (as an OLE *server*), or whether it will use or consume these services (as an OLE *client* or a *container*).

OLE Overview

OLE is a mechanism that allows users to create and edit documents containing data created by multiple applications. OLE documents seamlessly integrate various types of data, called *items*. Sound clips, spreadsheets, and bitmaps are typical examples of items found in OLE documents. Supporting OLE in an application allows

e user to work with OLE documents without worrying about switching back and forth etween the different applications; OLE does the switching.
container application is used to create OLE documents, and a server application is sed to create the items within the container application. Any application may be a ontainer, a server, or both.
LE incorporates many concepts that all work toward the goal of seamless interaction etween applications. These areas include the following:

eature	**Definition**
inking and mbedding	Linking and embedding are used for storing items inside an OLE document that were created in another application.
n-place activation	Activating an embedded or linked item in the context of the container application is called in-place activation. The interface of the container application changes to incorporate the features of the application that created the embedded or linked item.
Jniform data transfer	Uniform data transfer (UDT) is a set of interfaces that allows data to be sent and received in a standard fashion, regardless of the actual method chosen to transfer the data. UDT forms the basis for data transfers using the Clipboard and drag and drop.
Drag and drop	Drag and drop is an easy-to-use, direct-manipulation technique to transfer data between applications, between windows within an application, or even within a single window in an application. The data to be transferred is simply selected and dragged to the desired destination.
Compound files	Compound files provide a standard file format that simplifies structured storing of OLE documents for OLE applications. Within a compound file, "storage" have many features of directories, and "streams" have many features of files.
OLE Automation	OLE Automation allows one application to drive another application. The driving application is known as an automation client, and the application being driven is known as an automation server.
OLE Component Object Model	The OLE Component Object Model (COM) provides the infrastructure used when OLE objects communicate with each other. The Microsoft Foundation Class Library OLE classes simplify OLE COM for the programmer.

Visual Editing

Support for visual editing includes in-place activation and editing, drag and drop, nd OLE document (structured) storage. Two sets of MFC classes support visual diting. One set of classes is designed to help developers build OLE servers, the oftware components that can be embedded or linked inside other applications. This erver support lets developers visually export the components in their applications. The ther set of classes is designed to help developers build OLE containers that allow the

omponents from other applications to be visually embedded or linked in OLE locuments.

Visual Editing Servers

A visual editing server application can create OLE items for use by other pplications that have the appropriate OLE support to contain these items (for nformation on container applications, see the following section, "Visual Editing Containers"). Server applications usually support copying their data to the Clipboard or by drag and drop so that container applications can paste the data as an embedded or inked item.

A *miniserver* is a special type of server application that can be launched only by a container. The Microsoft Draw and Microsoft Graph servers are examples of miniservers. A miniserver does not store its own data as files on disk; instead, it reads its documents from and writes them to items in documents belonging to containers. As a result, a miniserver can only support embedding, not linking. A *full-server* can either be run as a stand-alone application or launched by a container application. A full-server can store documents as files on disk. It can support embedding only, both embedding and linking, or only linking. The user of a container application creates an embedded item by choosing the Cut or Copy command in the server and the Paste command in the container. Choosing the Copy command in the server and the Paste Link command in the container creates a linked item. When a server application is launched by a container application and is in place, handles all of the toolbar and menu negotiation to allow the server's menus and toolbar to display in place of the menus and toolbar of the container. Even advanced user-interface features such as docking toolbars are supported when a server is in-place activated; in this case, the toolbars are docked to the container's frame window. Handles all the negotiation with the container for the tool space, including showing correct feedback when the toolbar is being dragged.

Visual Editing Containers

A visual editing container application can incorporate embedded or linked items into its own documents. The documents managed by a container application must be able to store and display OLE items as well as the data created by the application itself. A container application allows users to insert new items or edit existing items by activating server applications when needed.Container applications will launch and contain a server application. Communication between containers and servers is achieved through the OLE system DLLs. This OLE system DLLs provide functions that containers and servers call, and the containers and servers provide callback functions that the DLLs call. Using this means of communication, a container doesn't need to know the implementation details of the server application. A container can accept items created by any server without having to define the types of servers with which the container can work. As a result, the user of a container application can take advantage of future applications and data formats. As long as these new applications are OLE servers, an OLE document will be able to incorporate items created by those applications.

Drag and Drop

Drag and drop is an easy-to-use, direct-manipulation technique to transfer data between applications, between windows within an application, or even within a single window in an application. The data to be transferred is simply selected and dragged to the desired destination. One application provides the data for copying, and another application accepts the data for pasting. Each side of the transfer needs to perform different operations on the data for the transfer to succeed. The library provides full support to represent each side of this transfer.
Data sources represent the source side of the data transfer. They are created by the source application when data is provided for a drag-and-drop operation. Data objects represent the destination side of the data transfer. They are created when the destination application has data dropped into it.

Compound Files (Structured Storage)

Compound files are an integral part of OLE. They are used to facilitate data transfer and OLE-document storage. Compound files are an implementation of the structured-storage model. Supports using both compound files and normal flat files for an application's file storage. The code the developer writes is the same. Using the **CArchive** abstraction, is able to hide the differences between these two file formats while offering the benefits of each. These benefits include support for serialization to storage (using the OLE **IStorage** interface), a stream (using the OLE **IStream** interface), or a normal or flat file where the developer has completes control over the file format.

OLE Automation Servers

An OLE Automation server exposes programmable software components, along with their properties and methods, to other applications. These driving applications are called OLE Automation clients. Exposing objects in this way is beneficial when applications provide functionality that is useful for other applications. For example, a word processor might expose its spelling-check functionality so that other programs can use it. Exposure of objects enables vendors to improve the functionality of their applications by using the ready-made functionality of other applications.
ClassWizard, AppWizard, and the framework all provide extensive support for OLE Automation servers. They handle much of the overhead involved in creating an OLE Automation server so that developers can focus their efforts on the functionality of their applications. Along with ClassWizard support, makes it extremely easy to expose member variables and member functions from type-safe C++ classes as OLE Automation properties and methods. This allows an application's objects to be driven by an external macro language such as Visual Basic. Provides direct support for creating type libraries. Type libraries are useful for exporting OLE Automation servers to other C++ client applications. The new AppWizard will automatically create an .ODL file, and

assWizard will automatically maintain the .ODL source file used to create the type rary.

LE Automation Clients

An OLE Automation client manipulates components implemented in other plications. The application being manipulated, which is the OLE Automation server, xposes programmable components and allows clients to automate certain procedures directly accessing the server components, along with their properties and methods. y creating an OLE Automation client, makes it easy to drive other applications, lowing developers to build an application using C++ language to drive the objects xposed by large applications such as Microsoft Excel, Microsoft Word, or any other LE Automation server that has a type library. ClassWizard reads the type library rovided by the OLE Automation server and creates new custom C++ classes for the **Dispatch** interfaces exposed by that server. provides the glue to connect these ynamic OLE (**IDispatch**) interfaces into type-safe C++ classes. Therefore, most OLE utomation clients written in C++ do not have to deal with the dynamic nature of **Dispatch**.

LE Controls

The OLE control architecture merges the popular VBX custom control rchitecture with the open, standard architecture of OLE. OLE controls make omponent-based development a reality by allowing developers to easily use existing odies of functional code encapsulated as OLE controls. An OLE control is a custom ontrol, implemented as an OLE object with visual editing and OLE Automation support. n OLE control has additional capabilities beyond those of ordinary OLE components, uch as the ability to fire events. With, using AfxEnableControlContainer supports ontainment of OLE controls. Most OLE objects require a substantial amount of mplementation effort. Fortunately, the OLE Control Development Kit (CDK)—built into isual C++ 4.0—provides most of the required implementation, so developers only have o fill in details that are specific to the OLE control. The OLE control CDK classes hemselves is based on base classes (that is, **COleControl** derives from **CWnd**). This nakes it easier for developers who are experienced with to create OLE controls. It also nakes it easier to use existing code inside the implementation of new OLE controls. ncludes support for creating and using 32-bit OLE controls. Along with the newly nerged CDK, provides support for creating, testing, and using OLE controls, including undreds available from third parties.

Architecture Classes

A key benefit of an application framework is that it not only provides a large body f prebuilt functionality but offers an architecture in which to add new functionality. An legant architecture gives developers a logical and obvious location to add application-specific code when implementing features in their applications. For example, when developers are implementing the File Save command, the application framework should

ave an obvious technique (such as a member function or a hook) to add this unctionality. It is not enough, however, to point programmers to the right place, ecause there is often no single right place, or the application framework may not oresee the exact situation. Addresses these issues with a group of tightly integrated lasses collectively known as the *application architecture classes*. These classes rovide support for the important areas common to many parts of the application, such s commands, documents, views, printing, online help, and dialog processing.
Commands Menu items, keyboard accelerators, and toolbar buttons are the most common sources of commands in an application. A *command* is an instruction to the program to perform a certain action. Unlike a procedure or function call, a command is a message that is routed to various command targets that may carry out the instruction. Command *targets* are objects derived from the **CCmdTarget** class, and they include documents, views, windows, and the application itself.
The command architecture ensures that any user-interface action, such as clicking a toolbar button or selecting a menu item, will route the command to the appropriate handler. Command routing can also be used to update the visual state of menu items or toolbar buttons. For example, the Edit Cut command might have both a menu item and a toolbar button that can be enabled or disabled. Using the command architecture, it is easy to maintain the visually enabled or disabled state of both the menu item and the toolbar button with a single line of code in a single location.type-safe mechanism for directing any windows message, control notification, or command to a C++ member function in the appropriate class. Each command target has a message map, and it contains entries that map each command ID defined in the Visual C++ resource editor to a C++ member function. Because there can be a large number of commands (as well as Windows-based messages and notifications) handled by each command target, ClassWizard is usually used to create classes and to maintain a class's message maps and message-handler functions.MFC extended the use of the command-handling architecture to support OLE Automation. Any class derived from **CCmdTarget** can expose member variables and member functions as OLE Automation properties and methods. MFC also included macros to make it easier to handle ranges of command IDs, a commonly requested feature.provides support for extended control notifications (WM_NOTIFY), routed just like any other command. This support is included for new services that are exploited in the release of future operating systems, and it is another example of being on the leading edge of support for new operating systems.

Documents and Views

The document/view architecture introduced in MFC is the basis for managing the storage and display of application-specific data. The **CDocument** class provides support for managing an application's data and an application will typically derive a new class from **CDocument** for each document type. The key feature of a document class is its ability to save a document object to a file for later use. The programmer's responsibility is to override the **Serialize** member function, which saves and loads application-specific data to and from storage. By implementing this function, automatically supports high-level commands such as File New, File Save, File Save As, and File Open. Does all the work of displaying a dialog box to gather information from

the user and managing the disk file. Although most documents are typically associated with disk files, the **CDocument** architecture is flexible enough to allow manipulation of data stored in other ways, such as in a database file, or to allow manipulation of data without any kind of stored representation. AppWizard provides several options for the initial skeleton of an application that incorporates database functionality in a variety of ways.Each document in a running application is attached to one or more views of that document. Views control the graphical display of the application's data on the screen. Programmers typically derive a class from the **CView** class and then implement the display code. A view represents the main area of a window on the screen and is a simple child window that can be manipulated with **CWnd** member functions. This usually involves implementing the **OnDraw** member function and writing the code that displays the data that is currently visible to the user. The **OnDraw** function replaces the low-level **OnPaint** handler of MFC with a high-level abstraction. After implementing **OnDraw**, the program automatically supports printing and prints preview. The **CView**-derived class is usually the best place to handle most of the commands and window messages that graphically manipulate the data. To support high throughput and fast updates, a number of APIs enable optimization of the drawing process to support most professional applications. It is also easy to have many views on the same document, and each view can be a different **CView**-derived class. For example, a splitter window will have one view for each pane.To coordinate documents and views,uses the helper class **CDocTemplate**. This class orchestrates the creation of documents, views, and frame windows in response to user input. One document-template object is created for each document type and is the glue that connects the document and view types. The application object maintains the document templates. Two document-template classes are supplied: one for multiple-document interface (MDI) and one for single-document interface (SDI). The significant differences between an MDI and SDI user-interface model are encapsulated in the document template and frame window classes.

Printing and Print Preview

By leveraging the document/view architecture, MFC is able to provide an application with device-independent printing. This means that the same code written for **OnDraw** in the **CView**-derived class can be used to draw on the screen and on the printer. When the user asks to print a document using the standard File Print command, calls the **OnDraw** member function with a special device context that is aware of the current printer and knows how to translate the screen display into appropriate printed output. also provides support for all the standard printing user-interface dialog boxes. In combination with the printing and document/view architectures, supports print preview functionality, which shows a reduced image of either one or two pages of a document as they would appear when printed on the currently selected printer. The implementation provides the standard user interface for navigating between pages, toggling between one- and two-page viewing, and zooming the display in and out to different levels of magnification. The ability to support print preview is an excellent example of the level of prebuilt functionality and the high level of abstraction in . The print preview feature represents several thousand lines of code in the application framework, but programmers only need to handle the display output code in the

nDraw member function of class **CView** and make sure that the File Print Preview enu command is available—the framework does the rest.

alog Data Exchange and Validation (DDX/DDV)

The capability known as *dialog data exchange* (DDX) introduced in MFC ovides an easy way to initialize the controls in a dialog box and gather input from the er. An associated mechanism known as *dialog data validation* (DDV) provides lidation of the dialog data. The heart of the DDX/DDV feature is the **oDataExchange** member function, which is called automatically by the application mework when data must be transferred and/or validated.
cause there are so many possibilities for exchange and validation (for example, the e of custom controls or the need for application-specific validation schemes), crosoft made the DDX/DDV architecture fully extensible. Developers can supply their n DDX and DDV functions and integrate them seamlessly with. In MFC , this echanism was used primarily for dialogs. Now, the DDX mechanism has been tended to work with data-bound record views as well as with normal dialogs and mple form views. The DDX/DDV architecture is tightly integrated with ClassWizard, ich enables developers to define all the necessary member variables and DDX/DDV utines without having to write any code.

ontext-Sensitive Help

Support for online and context-sensitive documentation is essential for most olications. Provides an architecture that makes it easy to incorporate the two most mmon types of help support in Windows-based applications. Help support includes a lp menu with the standard commands and provides architecture for the application mework to map from command or resource IDs to the various help contexts. Help ntexts are easily created in Visual C++, because every time a user-interface element created in the Visual C++ resource editor, a help context for that element is tomatically created. Help files (. HLP) are authored using standard authoring tools. hen a user presses the F1 key, automatically processes the keystroke as a help quest for the current command target. For example, the **CDialog** class processes the lp request by invoking WinHelp on the help topic for the currently displayed dialog x. If no help context is defined for the current command target, then the application mework automatically launches the default help. The **CFrameWnd**, **CMDIFrameWnd** d **CDialog** classes all provide handler functions for help support. Developers can add pport to any class that is a command target. When a user presses SHIFT+F1, otures the mouse and changes the cursor into the standard context-sensitive help rsor (arrow + question mark). When this cursor is displayed, clicking a user-interface ect tells the application framework to invoke WinHelp with the correct help context sed on the selected object. Provides a tool to manage the help-context information, ich associates user-interface elements with help contexts. In addition, AppWizard ovides much of the standard WinHelp-format file with prewritten information on all of standard commands. All that is needed is an editor capable of editing Rich Text rmat (RTF) text, such as Microsoft Word, to add application-specific information. In

this way, AppWizard, and the Visual C++ development environment work together to provide programmers most of their applications' help features automatically.

High-Level Abstractions

High-Level Abstractions was the cornerstone of a robust framework for building reusable classes, but it did not provide enough high-level abstractions to reduce programming time. With and later, this issue is addressed with a set of classes that support the most common user-interface idioms and provide capabilities for taking advantage of other rebuilt functionality. These classes, collectively called *high-level abstractions*, are designed to be used as supplied by and can result in a dramatic reduction in programming time. In a few lines of code, programmers can build a text-processing window that integrates seamlessly with other MDI windows, or they can change the base class to turn a view into a scrolling view. In addition to this power, all of these high-level classes are designed to be easily modified using C++ inheritance.

Form View

The functionality of **CRecordView** is built upon the **CFormView** high-level abstraction introduced in . **CFormView** supports many features that true form-processing applications require but which are not available in the native Windows dialog manager, such as scrolling, multiple forms for the same data, synchronous update, and printing.

A **CFormView** provides a view (a class derived from **CView**) based on a dialog resource that can be edited with the Visual C++ dialog editor. This view can be used to create form views with arbitrary Windows controls. The user can scroll the form view and tab among controls. The benefit of **CFormView** over standard dialogs is that **CFormView** objects integrate with the entire application framework architecture, providing automatic support for command handling and document management. A form view can also be an MDI child window.

EditView

CEditView is a simple, plain-text-editor view that has all the functionality of the standard Windows edit control. In addition, however, **CEditView** supports high-level functionality such as printing, find and replace, cut, copy, paste, and undo, as well as the standard File commands (Open, Save, and Save As). Of course, because **CEditView** is derived from **CView**, all of the architectural benefits described above apply. From a simple AppWizard-created application, programmers can use **CEditView** by simply creating a document template that uses **CEditView**; applications can have an MDI text editor without the programmers having to derive their own view classes.

Scrolling View

Most applications can show only a portion of their data files on the screen at a single time. The **CScrollView** class, which is another high-level view class derived from

Obsolete Features

With MFC's tradition of providing very good backward compatibility, most and programs will work with which a few very minor modifications. Very few features of previous s of are no longer supported in or will not be supported in the future. Obsolete features are in technologies that are replaced by OLE technology.

Support for OLE

Provided nine classes to support of OLE. These classes were obsolete which have been replaced with the OLE s of these classes. Because OLE is a functional superset of OLE and provides many more features, writing OLE applications is no longer recommended.

VBX Controls (16-Bit Only)

Microsoft Visual Basic introduced the concept of add-in components known as VBX controls. VBX control functionality is replaced by the new OLE controls model. Provides support for creating, testing, and using OLE controls.

Windows API Classes

Provides classes that simplify programming for Windows while at the same time permitting application developers to leverage both existing Windows-based code and programming experience. For the inexperienced programmer for Windows, the Microsoft Foundation Class Library simplifies Windows-based programming by providing prebuilt functionality for many standard programming idioms. These classes have evolved from the implementation, but backward compatibility has been maintained.

Standard Application Support

Encapsulates the standard application structure in an easily customizable application object. In addition to standard initialization, message processing, and termination, the **CWinApp** class supports idle-time processing of user-defined operations. Additional features of the **CWinApp** class include support for profile settings, context-sensitive help, File Manager drag and drop, shell registration for launching the application from File Manager, and other user-interface features. The **CWinApp** class frees the programmer from the details of the WinMain, LibMain, and WEP routines and provides a standard abstraction across Windows platforms.

Frame Windows

Along with an application object, most programs will use a standard frame window. Provides support for both the single-document interface (SDI) and the multiple-

/iew, supports views that scroll and views that are automatically scaled to the size of e frame window that displays them. By deriving from **CScrollView**, developers can d the ability to scroll or scale to their view classes. **CScrollView** manages window zes and mapping mode for graphics, manages special modes needed for OLE in-ace editing, and handles the automatic scrolling in response to user-interface actions ch as clicking the scroll bar.

plitter Window

In a splitter window, the window can be split into two or more separately rollable panes. A splitter control in the window frame next to the scroll bars allows the er to adjust the relative sizes of the panes. Each pane is a different view on the same cument. This type of user interface is useful, for example, when a user wishes to view th the beginning and end of a very long document on a single screen. Provides the gh-level class **CSplitterWnd** to support this user-interface model. The **CSplitterWnd** ass also supports the two most common types of splitters: dynamic and static. With namic splitters, the user can add or remove arbitrary split panes; static splitters have predefined number of panes. Each of the splitter pane's views can be the same class, each can be a different derived **CView** class. In all cases, the application framework tomatically manages all aspects of the user interface and standard Windows essages.

ontrol Bars

CToolBar, **CStatusBar**, and **CDialogBar** all derive from the common base class **ControlBar**. The **CControlBar** abstraction enables the implementation to reuse code nong these classes. **CControlBar** provides the functionality for automatic layout thin the parent frame window of the derived classes. **CControlBar** demonstrates the wer of a base class that provides a partial implementation that is completed in a ries of closely related derived classes.The **CStatusBar** class implements a row of xt-output panes, or indicators. The output panes are commonly used as message es and status indicators. Examples include the menu help-message lines that briefly escribe the selected menu command and the indicators for the keyboard states of um Lock, Scroll Lock, and Caps Lock. The **CStatusBar** class supports any number of nes and automatically lays them out based on the width of the contents. Each pane n have a customized style, including three-dimensional borders, pop-out text, sabled, and stretchy. The command architecture supports automatic menu prompt rings, and when using the Visual C++ menu editor to edit menus for applications, ogrammers can also define the prompt string for the menu item. When creating a new plication with AppWizard, developers can specify whether or not the application will ovide a status bar.The **CDialogBar** class is like a modeless dialog in that it easily pports any combination of Windows controls and is created from a dialog template ited with the Visual C++ dialog editor. Dialog bars support tabbing among controls nd can be aligned to the top, bottom, left, or right edge of the enclosing frame window. ne most common example of a dialog bar is the print-preview user interface.

document interface (MDI). Many of the common MDI commands and user-interface functionality, such as changing the menu bar that is based on the active window, are provided as prebuilt functionality by the framework. In addition, error-prone areas of programming for Windows, such as keyboard accelerators and implementation of default behavior are handled in a seamless manner by the application framework. The document-template class in applications that take advantage of the document/view architecture manages frame windows. A view is contained within a frame window (usually a **CFrameWnd** or a **CMDIChildWnd**).

Graphics/GDI

The device-context class, **CDC**, provided a simple Windows API wrapper. extended the **CDC** implementation to allow polymorphic implementations of device-context output functions. This enables a virtual-display context that allows applications to use the same drawing code to send output to the screen, a printer, a metafile, or a print-preview view. Provides a complete set of classes for drawing graphical objects and managing device contexts. These graphical object classes include the entire standard Windows objects, including pens, brushes, bitmaps, fonts, regions, and palettes. Several device-context classes are also supplied to make the handling of common Windows idioms (such as window repainting) simpler and less error-prone. The graphical objects are designed to free system resources automatically when they are no longer needed, which simplifies common object-ownership problems and enables an application to run safely in a resource-constrained environment.

Dialogs

Makes it easier to use dialogs within an application. The application framework manages many of the intricate details of Windows-based system-oriented dialogs automatically, including the handling of dialog-specific messages. Dialogs are handled with the **CDialog** class, which supports both modal and modeless dialogs. Programmers simply derive from a dialog class and customize it by overriding member functions and message handlers. This customization model is exactly like every other **CWnd**-derived class, which provides good programming consistency.

Controls

Controls are windows that are drawn in the client area of frame windows or as controls in a dialog box. Provides classes for all of the standard controls: static text, buttons, edit control, list boxes, combo boxes, scroll bars, handwriting controls, and user-defined child windows. Makes it easy for developers to derive their own child windows (including deriving from the standard Windows controls) and to customize the behavior of the windows using C++ inheritance and message maps. Adds classes that support Windows common controls.

eneral-Purpose Classes

The general-purpose classes give programmers a wide range of functionality esigned to take advantage of the powerful features of C++. These classes are ailable for programmers to develop the nongraphical portion of the application. In any respects, the general-purpose classes, together with the Windows API classes, e the building blocks for the entire application framework and provide fundamental nctionality to those classes as well as programmer-defined classes.

un-Time Type Information

Most classes are derived either directly or indirectly from the class **CObject**, nich provides the most basic object-oriented features of the framework. **CObject** upports dynamic type checking, which allows the type of an object to be queried at run me. This feature provides programmers with a type-safe means to cast down a pointer om a base class to a derived class. Without dynamic type checking, this cast can be a ource of errors and can break the type safety of C++. Most programmers find this ature useful, but because it incurs a very small run-time overhead (approximately 24 tes per class), its use is optional.

bject Persistence

Persistence is the ability of any object to save its state to a persistent storage edium, such as a disk. If a collection is made persistent, then all members of that llection are made persistent. The **CArchive** class is used to support object ersistence and allows type-safe retrieval of object data. To use persistence, a class mplement must override the **Serialize** member function, call the base class' **Serialize** nction, and then implement the data-storage routines for member data that is specific a derived class. Entire networks of objects, with references to other objects, including oth multiple and circular references, can be saved with a single line of code. As with ynamic type checking, the use of persistence is optional.

ollection Classes

Excels in the efficiency of standard data structures. The provided collection lasses, a standard component of any C++ class library, are well tested, well coded, nd highly reusable. The collection classes include double-linked list classes, map dictionary) classes, and dynamic (growable) array classes. All of these have been mplemented using the proposed ANSI template syntax for type-safe usage. For xample, the list class is supplied with variants supporting **UINT**, **BYTE**, **WORD**, **WORD**, **void***, **CObject***, and **CString** elements. The map and array classes have imilar sets of variants. In all, supplies 17 collection classes. For users who wish to take dvantage of the template syntax to generate a type-safe variant of a supplied

implementation (or write their own templates), a template-expansion tool written using is provided as a sample application.

Strings

The **CString** class supports a very fast string implementation that is compatible with standard C **char*** pointers. This class allows strings to be manipulated with syntax similar to the Basic language that includes concatenation operators and functions such as **Mid**, **Left**, and **Right**. **CString** also provides its own memory management, freeing the programmer from having to allocate and free string memory.

Files

Offers three general-purpose file classes: **CFile** and its two derived classes, **CStdioFile** and **CMemFile**. **CFile** supports low-level binary file I/O (read, write, and seek). **CStdioFile** provides buffered file I/O similar to the standard I/O run-time libraries. **CMemFile** supports file semantics in RAM-resident files for managing Clipboard data as well as other forms of interapplication communication. The polymorphism provided by the three file classes (**CFile**, **CStdioFile**, and **CMemFile**) allows the same code to be used for sending data to a variety of destinations using the **CFile** interface. Extend the use of the file classes to include compatibility with OLE structure storage (**IStream** and **IStorage**).

Time and Date

In addition to the standard time and date functions, a class is provided to conveniently support time-and-date arithmetic using overloaded operators. Binary time values are automatically formatted into human-readable form.

3.3 HARDWARE REQUIREMENTS

Computer/processor

Personal Computer with a Pentium-class processor; 166-megahertz or higher processor recommended.

Memory

32 megabytes (MB) of RAM or 64 MB or more recommended.

Hard Disk Space

2 gigabytes (GB) or more recommended.

Display

VGA or higher-resolution monitors; Super VGA recommended.

Operating System

Windows NT workstation 4.0 or server 4.0 with service pack 4 or latter.

Peripherals/Miscellaneous

Microsoft Mouse or compatible pointing device.

RS 232C Cable for serial communication.

BAPCON (BHEL's Advanced precipitator controller)

BAPCON is a sub controller specially designed for ESP., which is one of the most sophisticated power controllers available.
The controller utilizes 8088 [Intel] family microprocessor component and support hardware. These components support have a proven reliability in control instrumentation in industrial and utility application and environment. Field programmability of operating parameter permits an extremely high degree of flexibility. The controller automatically selects and optimizes precipitator electrical operation based on field parameters.

Features of BAPCON

- Effective spark rate control.
- Deletion of spark / arc by dv/dt.
- Automatic current control based on steps all ram control settings.
- Intermittent charging technique.
- Measurement of peak, mean &valley of secondary voltage.
- Base change setting & measurement.
- Automatic selection of change ratio based on VI characteristics of the ESP.
- Annunciation of warning and trip alarms.

- Facility of remote control through 10mA balanced current loop.

OBJECTIVES

- Utilization of state of the art lip technology components ,H/w & S/w,
- S/w development specification for precipitator operation of the transformers-rectifier to actual real time precipitator operation condition
- Fast response to sparks
- Separate control strategy.

RAPCON

RAPCON is a microprocessor based unit that controls and surveys the operation of rapping motion is ESP. one RAPCON unit can control up to 16 rapping motors RAPCON starts and stops the rapping motors as programmed and will give an alarm if a rapping motor fails. RAPCON is a component of ESP-IOS. The RAPCON is pre-programmed with number of sequences, any of which can be selected depending up on the field failure condition of the ESP-IOS.

SYSTEM DESIGN

4. SYSTEM DESIGN.

System design is a process through which requirements are translated into a representation of software. It is the process of planning a new system to replace the old. Initially the representation depicts a holistic view of the software. Subsequent refinements lead to a design representation that is very close to the source code. The system design takes care of the following:

- Identification of outputs and reports of the new system.
- Scrutinize the data present on each report/output.
- Sketch of the form or display as expected to appear at the end of the completion of the system.
- Description of the data to be processed or stored.
- Individual data items and calculation procedures are written in details.
- The procedures written should tell how to process the data and produce the output.

4.1 ARCHITECTURAL DESIGN

System architecture is the high level part of software design, the frame that holds the more detailed part of the design. System architecture is also known as "System Design". The quality of the architecture determines the system. Good design makes implementation easy.

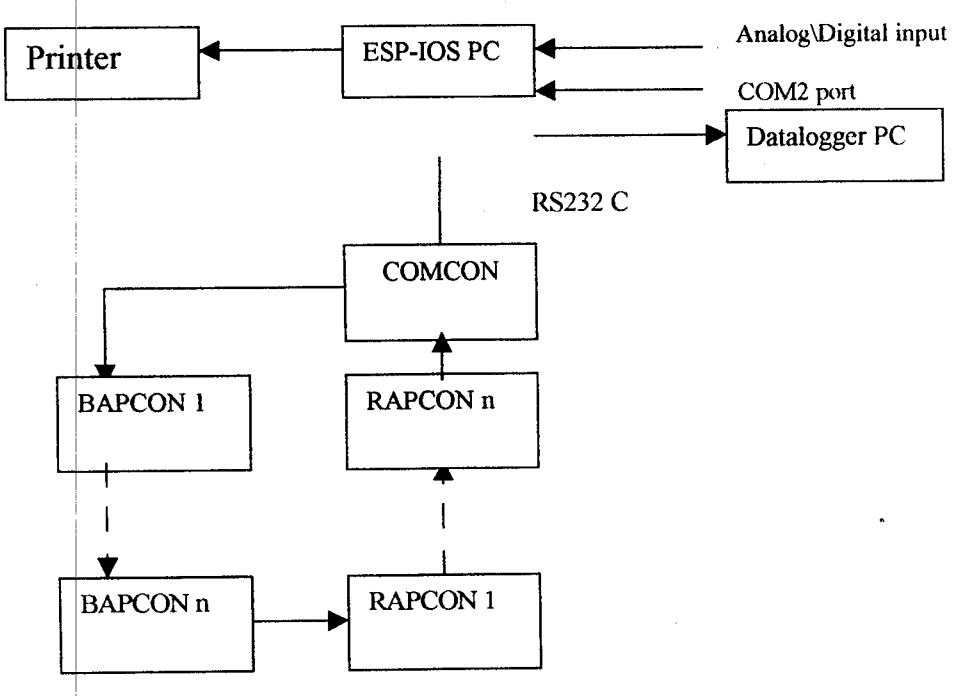
System architecture first needs an overview that describes the system in broad terms. We should find evidence that alternatives to the final organization were considered and find the reasons the organization used was chosen over the alternatives. The architecture should define the major modules in a program. A module is collection of routines that work together to perform a high-level function such as formatting output, interpreting commends, getting data from files or accessing major data structures.

The interface of each module should be defined. The architecture should describe which other modules the module could call directly, which it can call indirectly, and which it should not call at all. It should specify the data that module passes to and receives from other modules.

The architecture should describe a strategy for handling changes clearly. It should show that possible enhancements have been considered and that the most likely enhancements are also the easiest to implement.

The entire architecture is shown below.

Hardware Interface and Software Interface with other systems



4.2 INPUT DESIGN

The design decisions for handling inputs specify how data are accepted for computer processing.

Objectives for Input Design:

The quality of the system input determines the quality of the system output. Input specifications describe the manner in which data enter the system for processing. Input Design features can ensure the reliability of the system and produce results from accurate data or they can result in the production erroneous information. The input design also determines, whether the user can interact efficiently with the system.

Six objectives guiding the design of the input are as follows:

- **Effectiveness** : This means that input forms and screens serve specific purpose.
- **Accuracy**: Refers to design that assures proper completion.
- **Easy to use:** Screen designs for user interaction are straightforward and require no extra time to understand.
- **Consistency**: Means that forms and screens should group data of similar nature together.
- **Simplicity**: Refers to keeping the forms in the screen simple and uncluttered.
- **Attractiveness**: Input forms should be of appealing design that pleases the user.

All these objectives are important and can obtained by the use of basic design principles.

In our system input design is nothing but giving command to the BAPCON and RAPCON. They are,

MAIN MENU

When the ESP IOS program is executed, the opening screen will display the following menus.

This menu consists of the following sub menus.

... for commands and settings of sub controllers.

... for status display of sub controllers.

COMMANDS

Commands to the sub units BAPCON, RAPCON, etc are set using this menu.

The following sub menus are available.

- Commands and settings of BAPCON.

- Commands and settings of RAPCON.
  Depending upon the project configuration, other sub units like Statcon-E etc will appear in this command menu.

Auto Mode:

This is a command to execute sets of Closed Loop to optimize the ESP performance.

Current mode IOS (Manual/Auto) is indicated in the status bar of the main form.

Exit:

To exit IOS program.

BAPCON COMMAND

In this form, the default values are indicated near to command legend. Next to that suitable controls are provided for the user to give command to the BAPCON. Press SEND key after entering values to transmit the command. The default values displayed in the form menu are either last command sent or local values / status.

A status indication about the communication link is provided. Red indicates communication is not ok. Green indicates communication is ok.

The list of commands that can be given to BAPCON from IOS is indicated in the table given below. Also the range of input values and error message also given in the table.

BAPCON PARAMETERS

ITEM	RANGE	ERROR MESSAGE
BAPCON NO.	1 to Max. no of BAPCONs connected.	UNIT NUMBER OUT OF RANGE.
T/R POWER	ON/OFF	-
OPTIMISE	ON/OFF	-
CHARGE RATIO	1to 159	CHARGE RATIO SHOULD BE ODD NO. CHARGE RATIO OUT OF RANGE.
AV.CT.LIMIT	*0to Max. Rating of T/R set.	CURRENT LIMIT OUT OF RANGE.
IC.CT.LIMIT	0 to Max. Rating of T/R set x 2.	CURRENT CHARGE OUT OF RANGE
BASE CHARGE	0 to 49	BASE CHARGE OUT OF RANGE
REPEAT TIME	0.1 to 25.5	REPEAT TIME SETTING OUT OF RANGE.
ALARM	OFF	-

* In 1:1 charge ratio.

In case of intermittent charging,

Average Current Limit = 0 to (Max. current rating of T/R set x 2) / Charge Ratio)

RAPCON COMMAND

On display of this form, the default values of time factor and set number for rapcon-1 is indicated near to command legend. Next to that suitable controls are provided for the user to enter command to the RAPCON. Press SEND key after entering values to transmit the command. Similarly Time factor, Set number can be given to other Rapcons also.

The default values displayed in the form menu are either Last command sent or Local values / status.

ITEM	RANGE	DEFAULT	ERROR MESSAGE
RAPCON NO.	1 to Max no. RAPCONs connected		--
TIME FACTOR	0.0 to 25.5	Last sent values or as in LOCAL. Mode	RAPPER time factor out of range.
SELECT SET NO	1 to Max. no of SETs available..		RAPPER set number out of range

Further to these commands, following commands are provided through separate forms.

A status indication about the communication link is provided.
Red indicates communication not ok. Green indicates communication ok.
Also a message box displays the error status if a command is given.

RAPCON COMMAND - AUTO ON / OFF

Select the motors to be made Auto On/Off. Selected motor is shown in black color with / without tick mark.
Press SEND key to transmit the command.

Motors selected in AUTO MODE are shown highlighted in pink color based on the reply from respective Rapcon.

Settings, which were changed, can be cancelled by pressing CANCEL key.

RAPCON COMMAND - MANUAL ON / OFF

Select the motors to be made Manual On/Off. Selected motor is shown in black color with / without tick mark.
Press SEND key to transmit the command.

Motors selected in Manual MODE are shown highlighted in pink color based on the reply from respective Rapcon.

Settings, which were changed, can be cancelled by pressing CANCEL key.

RAPCON COMMAND - GET RAPPER TIME

This form displays the Rapper timings of the set number entered.

This timings are the preprogrammed timings of the respective set no. and cannot be altered. However, when this set is selected using "SET
RAP. TIME" command, the set '0' (Operating Set) is copied with this set. This can be edited as described in the SET RAPPER TIME section.

If the RAPCON is not connected to the system an empty table will be displayed, with error annunciation.

Settings which were changed can be cancelled by pressing CANCEL key.

Set Rap Time

This form displays the present rapping frequency, on which the RAPCON is working. The present rapping frequency can be modified using this form and sent to Rapcon by pressing SEND key.
In case of power supply interruption to RAPCON and when power supply resumes the standard rapping frequencies as per set no.1 will be transferred to set number-0 and used by RAPCON automatically. If the RAPCON is connected to the system an empty table will be displayed with error annunciation.

If the RAPCON is not connected to the system an empty table will be displayed, with error annunciation.

Settings, which were changed, can be cancelled by pressing CANCEL key.

Set Real Time

Use this form to alter required date and time in all RAPCONs.
Date and Time controls provided in this form can be edited to the requirement and press SEND key to transmit new date and time to the RAPCON.

If the RAPCON is not connected to the system error annunciation will happen.

Settings, which were changed, can be cancelled by pressing CANCEL key.

Alarm Reset

This form is meant for resetting the alarm of a rapper connected to a Rapcon.

Select the motors to be made Alarm Reset. Selected motor is shown in black color with round mark.

Press SEND key to transmit the command.

If the RAPCON is not connected to the system error annunciation will happen.

Settings, which were changed, can be cancelled by pressing CANCEL key.

4.2 OUTPUT DESIGN

One of the most important features of the system is the output it produces. Without quality output, the entire system may appear to be unnecessary that users will avoid using it. Users generally merit the system solely by its output. Therefore an effective output design is an important feature of design specification.

Objectives of Output Design:

Since useful output is essentially to gaining use and acceptance of the system, the systems analysts should try and following objectives which are for designing acceptable outputs.

- Design the output to serve the intended purpose.
- Design output to fit the user.
- Deliver the appropriate quantity of output.
- Assure that the output is obtained, where it is needed.
- Provide output on time.
- Choose the right output method.

DISPLAY

Status display of sub units BAPCON, RAPCON, etc can be viewed using this menu.
The following sub menus available are.

STATUS DISPLAY BAPCON

The STATUS DISPLAY for BAPCON allows viewing of the status of BAPCONs. The status of up to a maximum of 64 BAPCONs can be displayed depending on the project configuration.

The parameter, which appears in this display, is the status available in the respective BAPCONs, except OPTM. REACHED (AUTO) and FIELD SYMBOL.

OPTM. REACHED (AUTO):

If the OPTIMISER ALGORITHM is selected and when IOS is in auto mode the optimum reached feedback will be displayed as YES in this status display. During this operation OPTM. FEEDBACK is ignored.

FIELD SYMBOL:

The symbol of the BAPCONs allotted as per the site requirement. See SETUP.

STATUS DISPLAY RAPCON

The STATUS DISPLAY for RAPCON allows viewing of the rapper motor status of RAPCONs. The status of up to a maximum of 16 rappers and RAPCONs to a maximum of 8 depending upon the project configuration can be viewed depending on the project configuration.

For all rappers of a RAPCON, Auto/Manual/Running/Error status of motor is indicated by means of glowing LED display. In addition to that following status is displayed.

- RAPCON No.
- Rapcon Symbol
- Time Factor
- Set No.
- Status (Local/Remote)

SYSTEM IMPLEMENTATIO

5. SYSTEM IMPLEMENTATION

System implementation includes running, testing and the use of the system. After completion of the system analysis, design and coding, the user and the management evaluates the new system to ensure that it fulfils all its goals.

Thus implementation is the stage of project where the critical design is turned into a working system. The system can be introduced in two methods namely parallel running and the other is the sudden implementation [tested along with the existing system]. The former one is a good method for smooth transfer of the old system to the new system.

But here there is a repetition of work, which is being done by processing the current data by both the old and the new system, which is kept alive and operational until the new system has been provided for at least one system cycle.

In the second method the old system is suddenly scrapped down and the new system is introduced all of sudden. This reduces the duplication of work. But here the drawbacks of the new system may lead to confusion and total disorder of the system.

The new system is implemented by using the parallel running, thus the system is going to be implemented simultaneously along with the current system for testing.

The process that is done during the implementation is checking the proposed system with the existing system.

TESTING

6. SYSTEM TESTING

Software testing can be considered as the combination of two elements – Verification and Validation. Verification refers to the set of activities that ensures the software correctly implements a specific function. Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements. In simpler terms it can be explained that verification ensures that the product is being developed in the right way and validation ensures that the right product is being developed.

Any software has to under go series of tests during its development stage. The main types of tests that any software has to undergo are,

- Unit Testing.
- Integration Testing.
- Validation Testing.

6.1 Unit Testing

This testing focuses on each module individually, that it functions properly as a unit. It also ensures that all the paths in the module's contract structure are covered. This can also be used as a tool to uncover errors which may arise due to improper or inconsistent typing, erroneous initialization or default values, incorrect

or miss spelt variable names, inconsistent data types, errors that may occur during file handling.

Hence the individual modules were first tested for their effective performance. The system was tested with sample data and the irregularities that occurred were corrected and again put into testing.

6.2 Integration Testing

After the successful completion of unit testing, the integration testing is performed. The integration testing is systematic technique for construction of program structure at the same time conducting tests that uncovers errors associated with **interfacing.** This testing can be used as a tool to uncover – errors like of data across interface, the adverse effect that one module might have on another, problems rising due to global data structures.

When individual modules were found to work satisfactorily the system integration was performed and testing was carried out. All the required outputs were generated and corrections were made till they were found to be satisfactory.

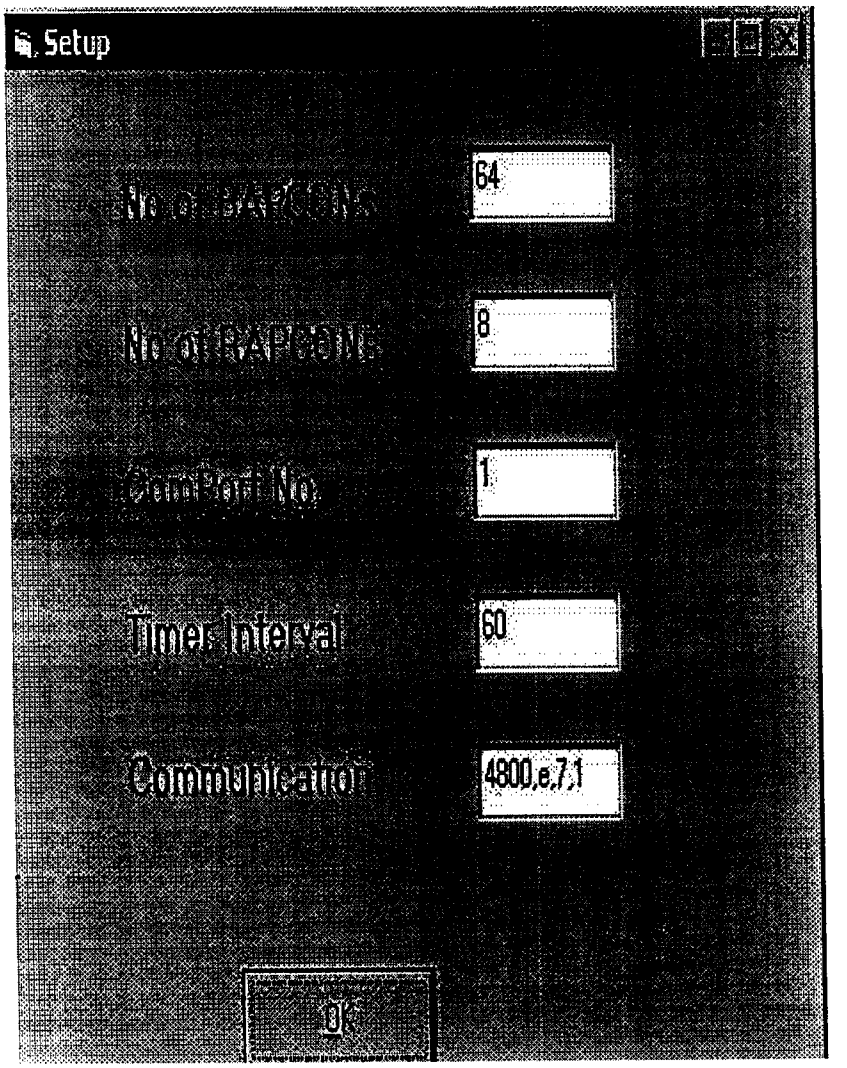
6.3 Validation Testing

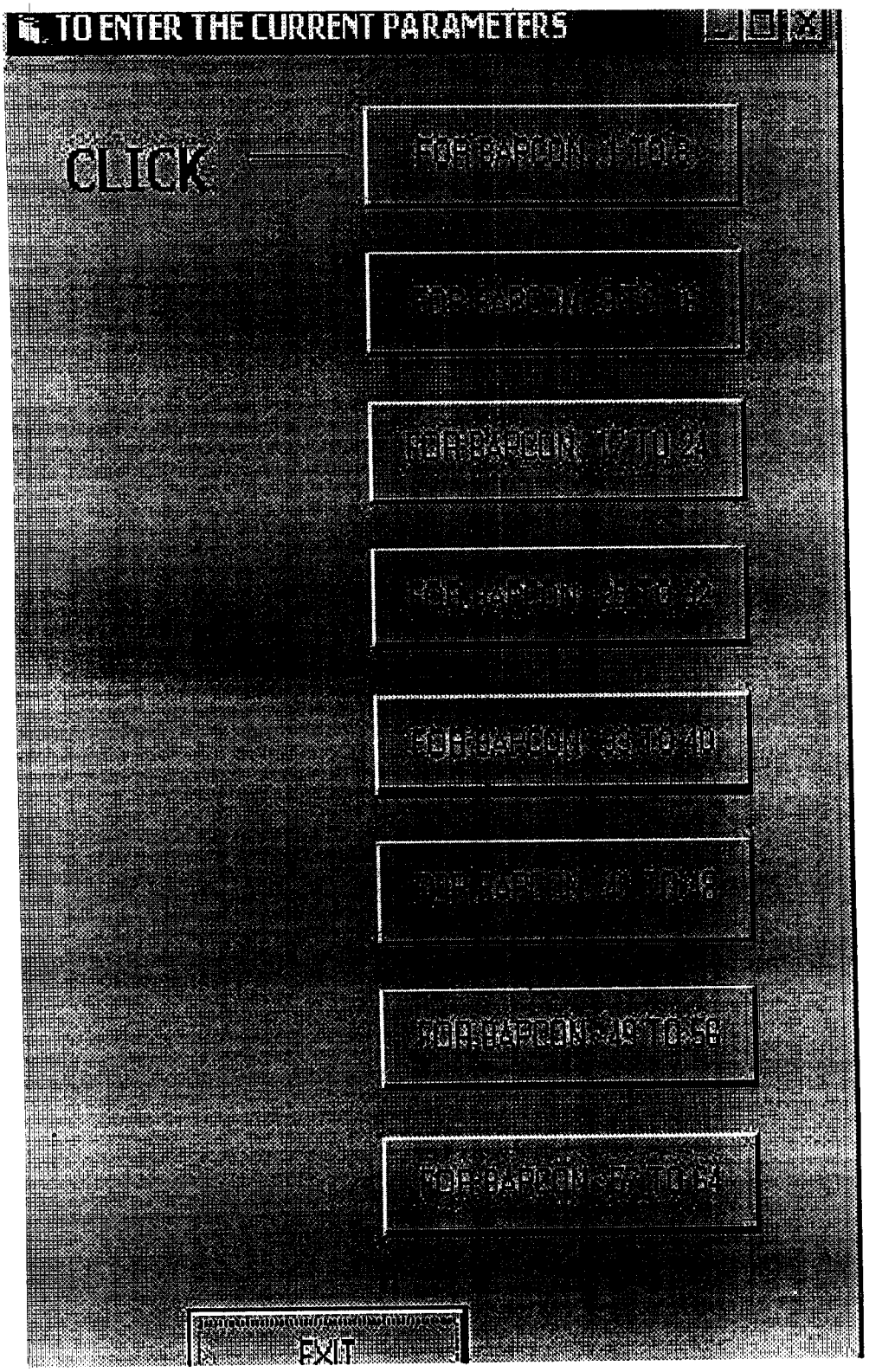
At the culmination of integration testing when the software has been completely assembled and all known errors have been uncovered and corrected, the validation testing is carried out. Validation is the set of activities that ensure that the software functions in a manner that is expected by the customer. Hence, validation testing ensures that all functional requirements are satisfied and all the performance requirements are satisfied.

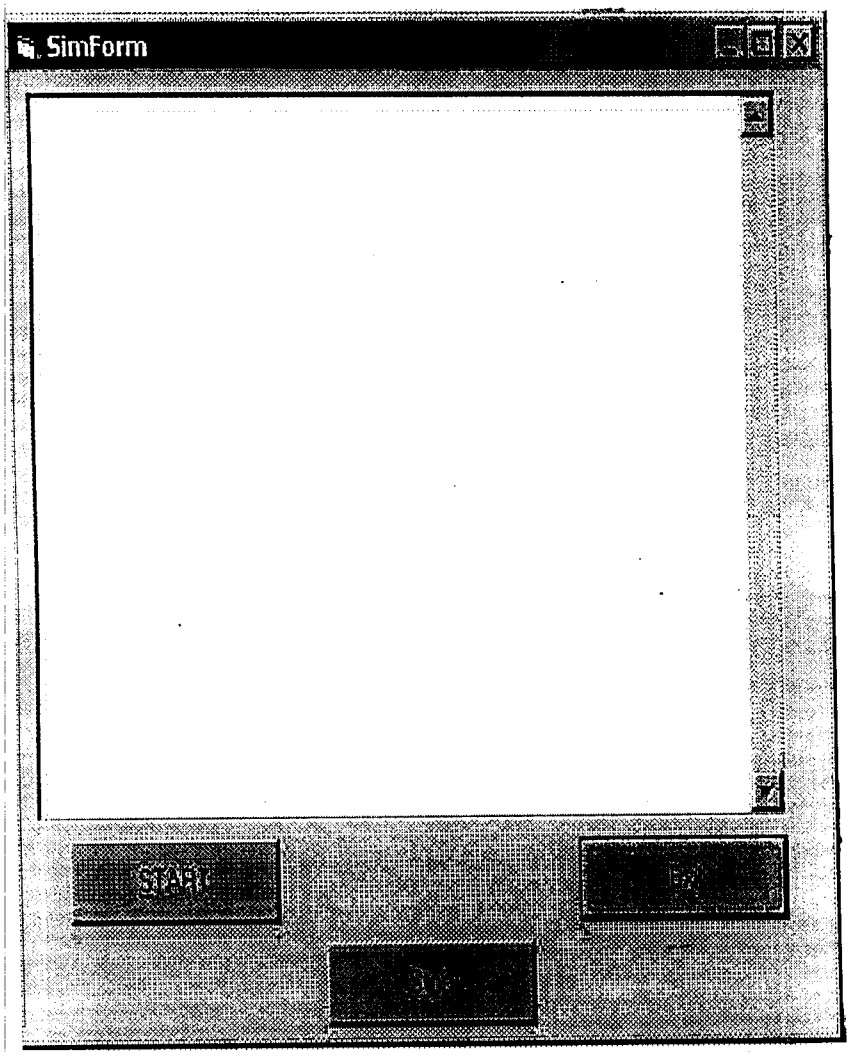
Hence, validation testing was carried out to the confirmation that the function performed according to the specification.

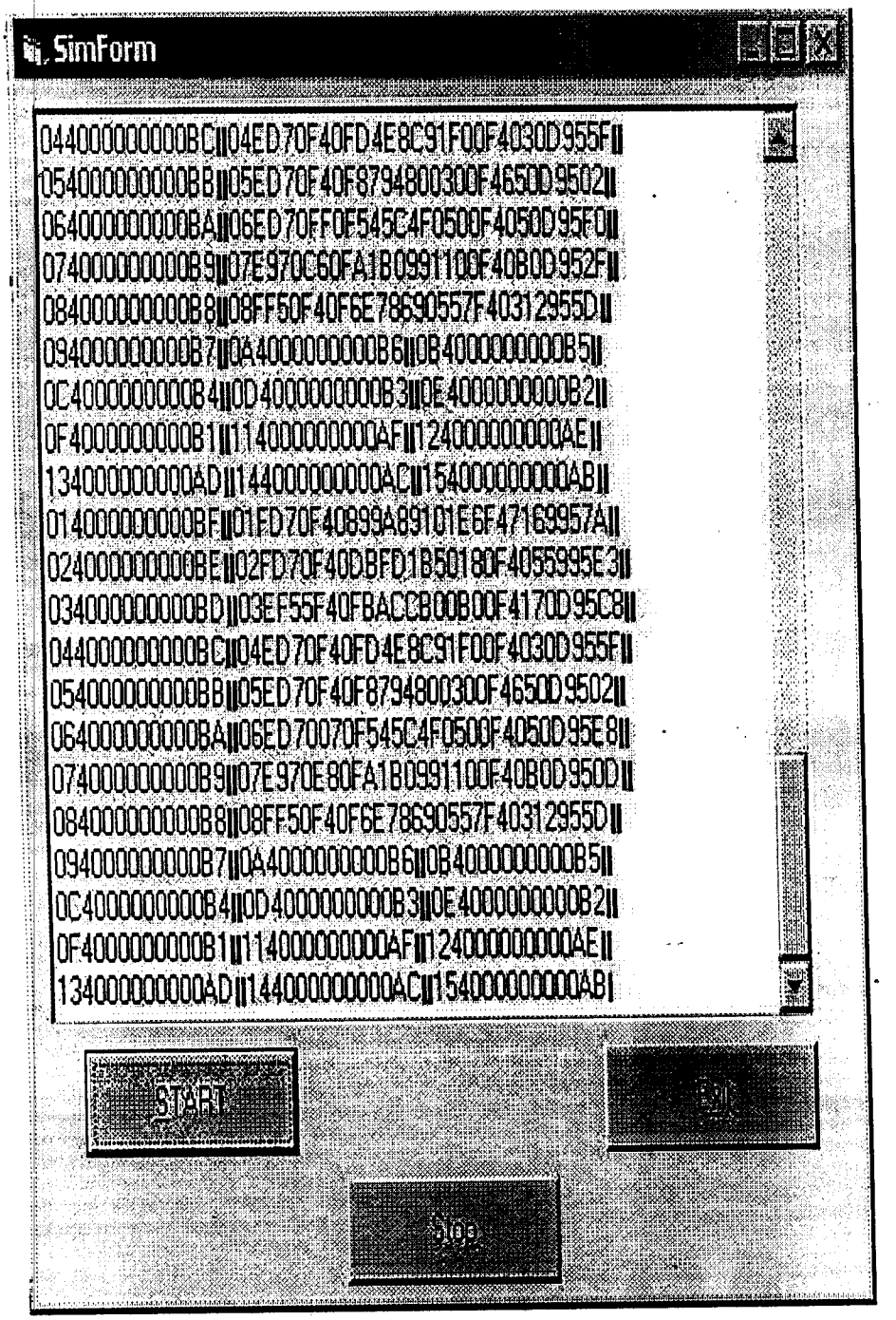
At various stages, the system is subjected to undergo all the above-mentioned tests and it is found to be working satisfactorily.

In this application the validity test was done with testing software named SIMULATOR specially designed for ESP-IOS.


Setup
64
8
1
60
4800,e,7,1
OK


TO ENTER THE CURRENT PARAMETERS
CLICK
EXIT


SimForm
START

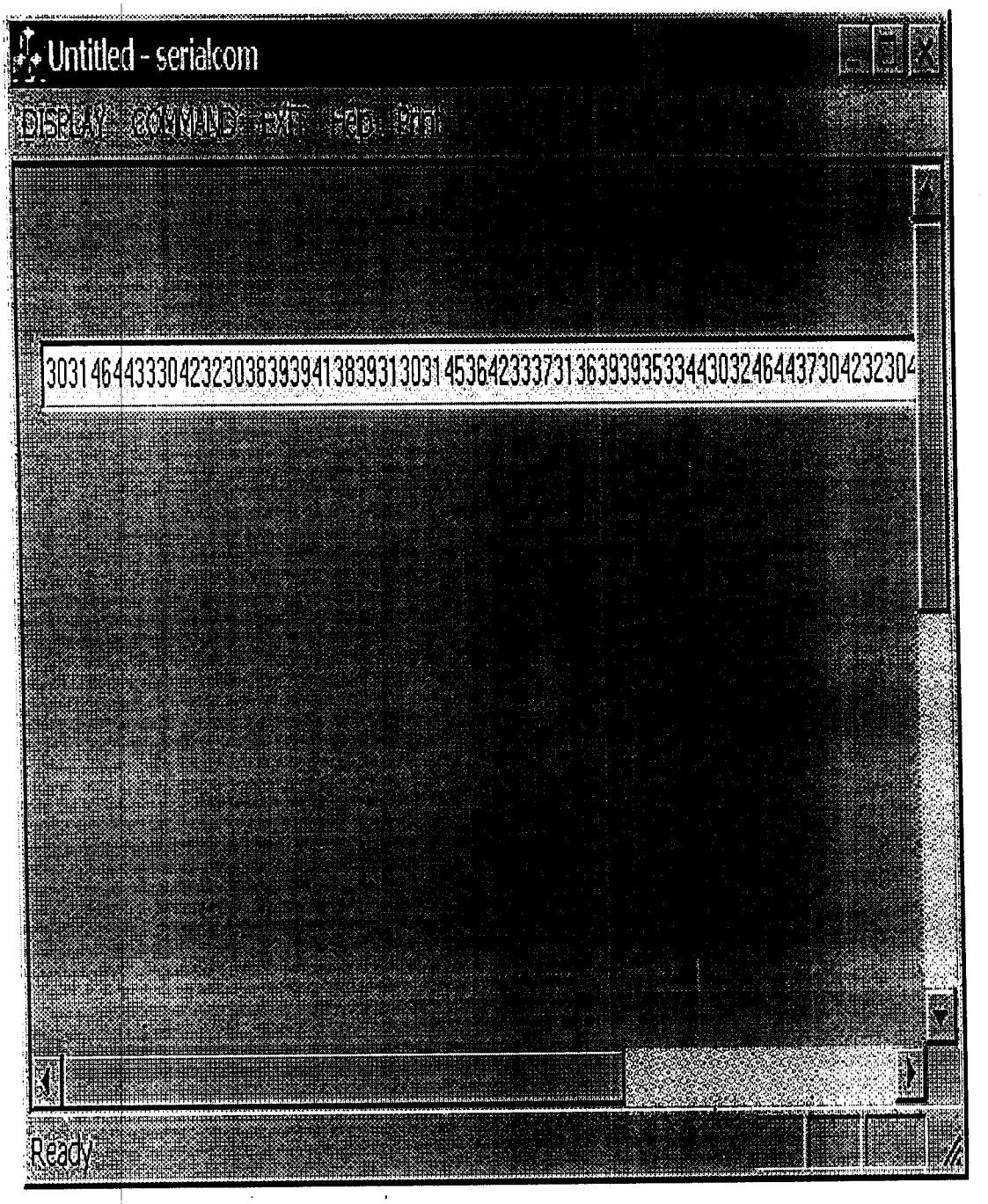

SimForm
044000000000BC||04ED70F40FD4E8C91F00F4030D955F||
054000000000BB||05ED70F40F8794800300F46500D9502||
064000000000BA||06ED70FF0F545C4F0500F4050D95F0||
074000000000B9||07E970C60FA1B0991100F40B0D952F||
084000000000B8||08FF50F40F6E78690557F40312955D||
094000000000B7||0A4000000000B6||0B4000000000B5||
0C4000000000B4||0D4000000000B3||0E4000000000B2||
0F4000000000B1||1140000000000AF||1240000000000AE||
134000000000AD||144000000000AC||154000000000AB||
014000000000BF||01FD70F40899A89101E6F47169957A||
024000000000BE||02FD70F400BFD1B5018DF4055995E3||
034000000000BD||03EF55F40FBACCB00B00F4170D95C8||
044000000000BC||04ED70F40FD4E8C91F00F4030D955F||
054000000000BB||05ED70F40F8794800300F46500D9502||
064000000000BA||06ED70070F545C4F0500F4050D95E8||
074000000000B9||07E970E80FA1B0991100F40B0D950D||
084000000000B8||08FF50F40F6E78690557F40312955D||
094000000000B7||0A4000000000B6||0B4000000000B5||
0C4000000000B4||0D4000000000B3||0E4000000000B2||
0F4000000000B1||1140000000000AF||1240000000000AE||
134000000000AD||144000000000AC||154000000000AB||
START
Stop

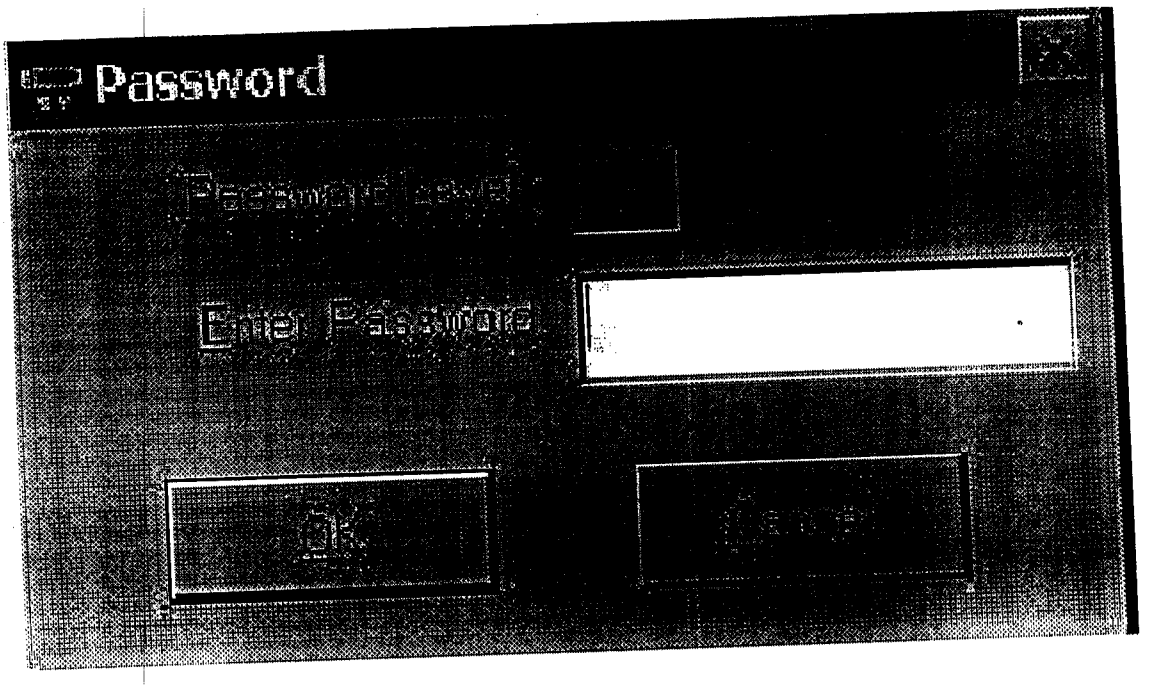
FUTURE ENHANCEMENT

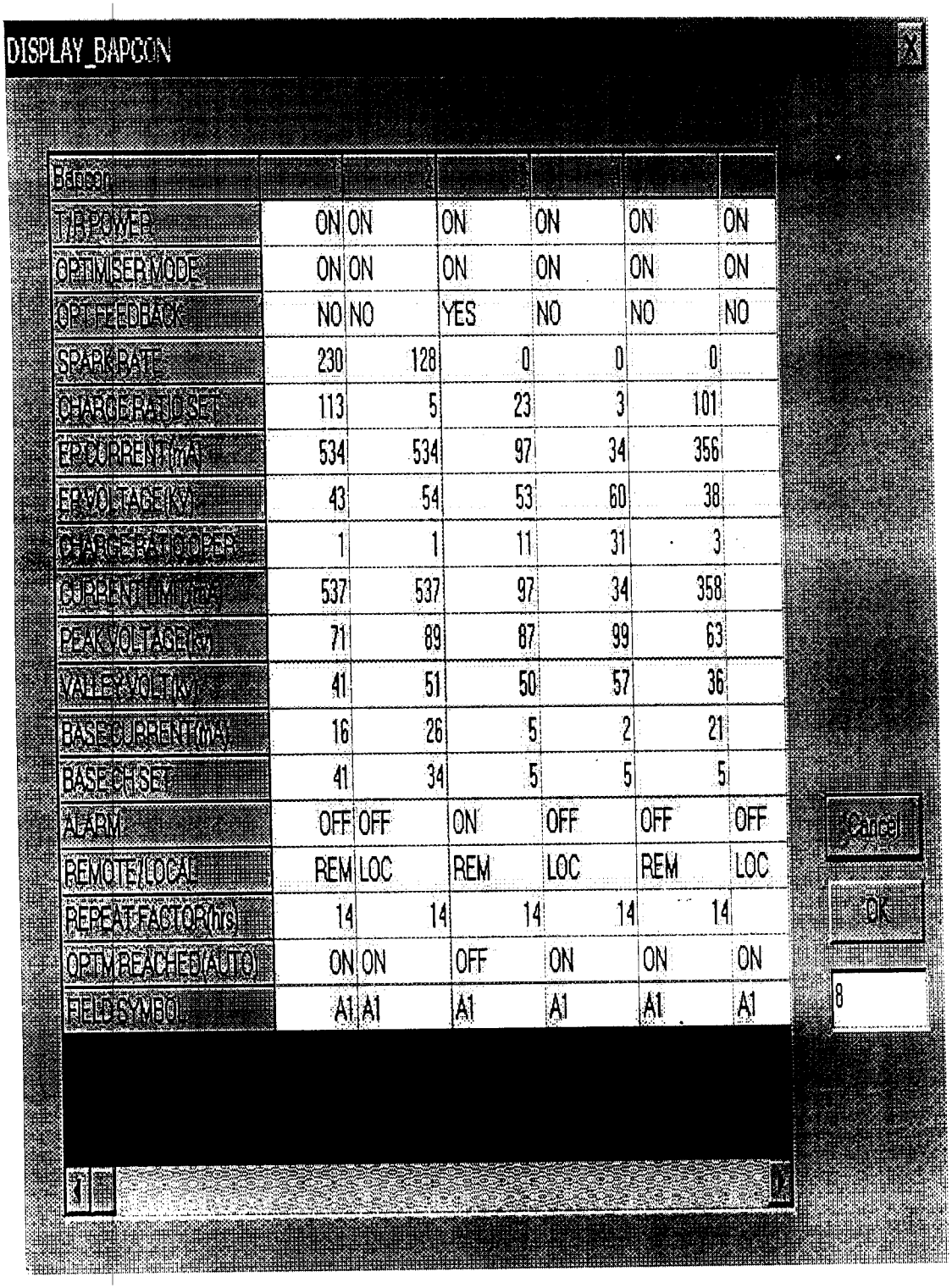
7. FUTURE ENHANCEMENT

- Analog display of BAPCON and RAPCON could be established.
- Separate algorithms could be generated and attached so that certain job could be done regarding ESP. (for example, algorithm resetting)
- Performance Evaluation of the entire system could be monitored.
- Analog input to the system could be given by having an Analog to Digital converter card.
- Graphical representation of data could be view with the help of Graphics server. Like VI curve etc.
- Reply from BAPCON and RAPCON can be view as a web page for plant manager to view the status.

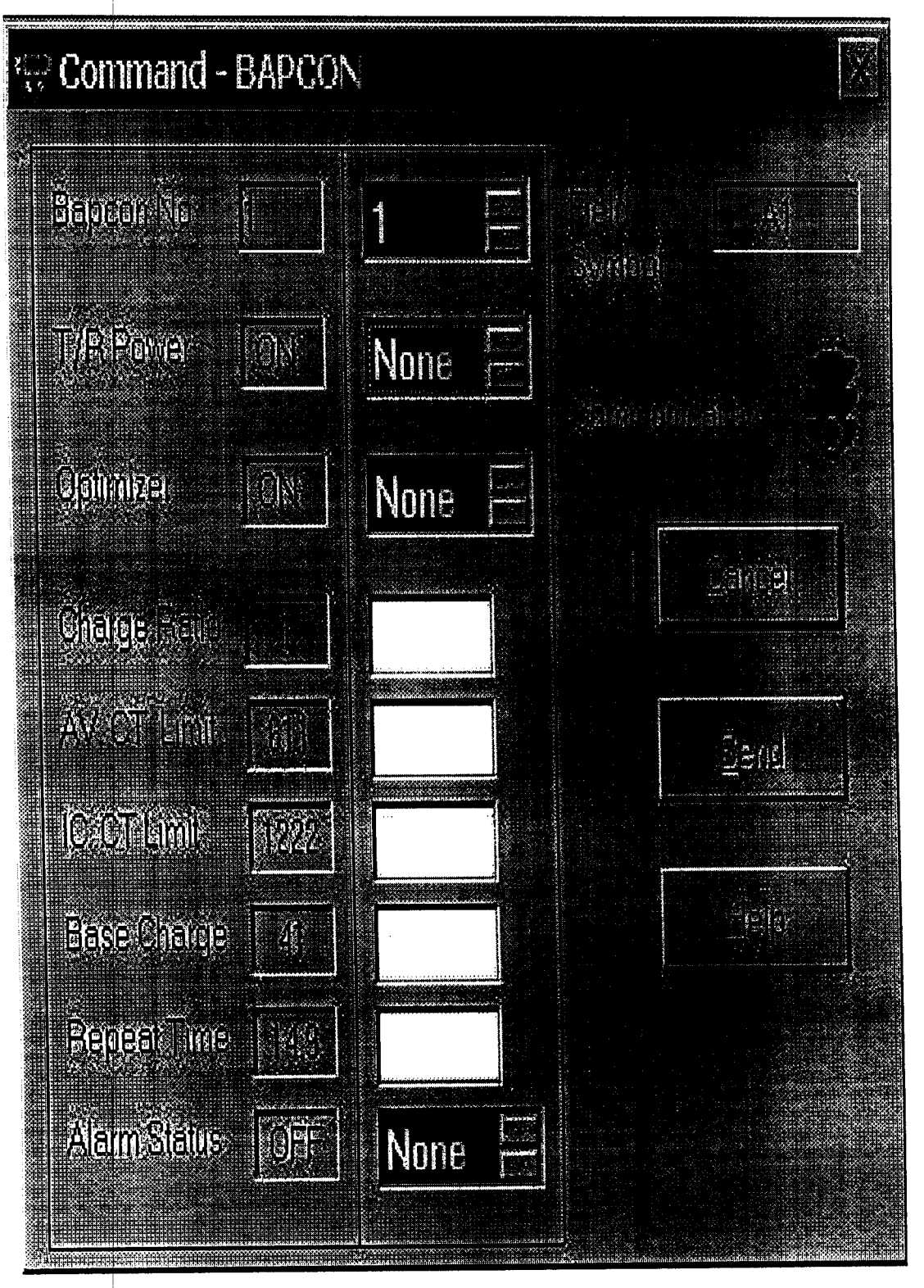
APPENDICES

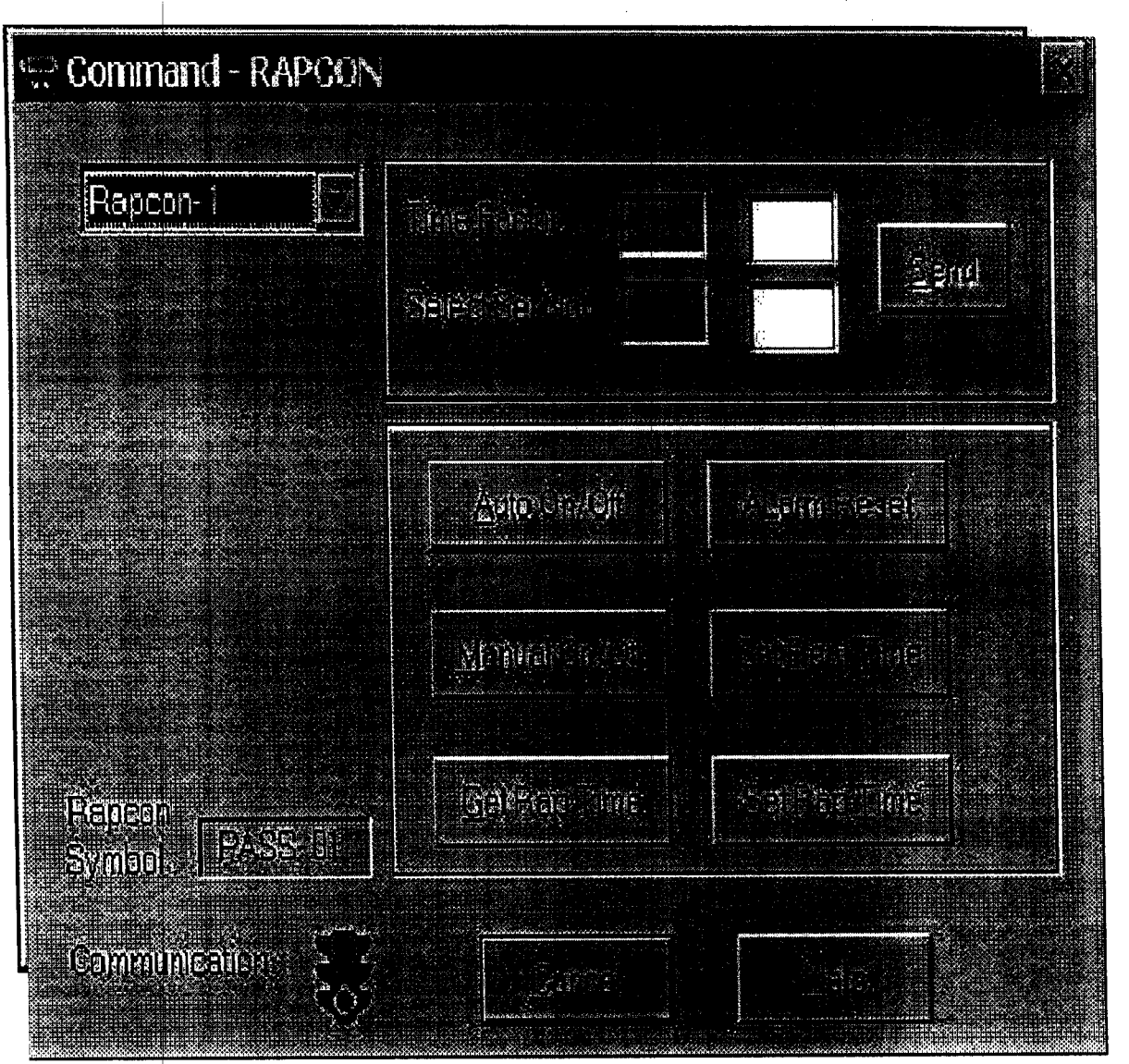

Untitled - serialcom
3031464433304232303839394138393130314536423337313639393533443032464437304232304
Ready

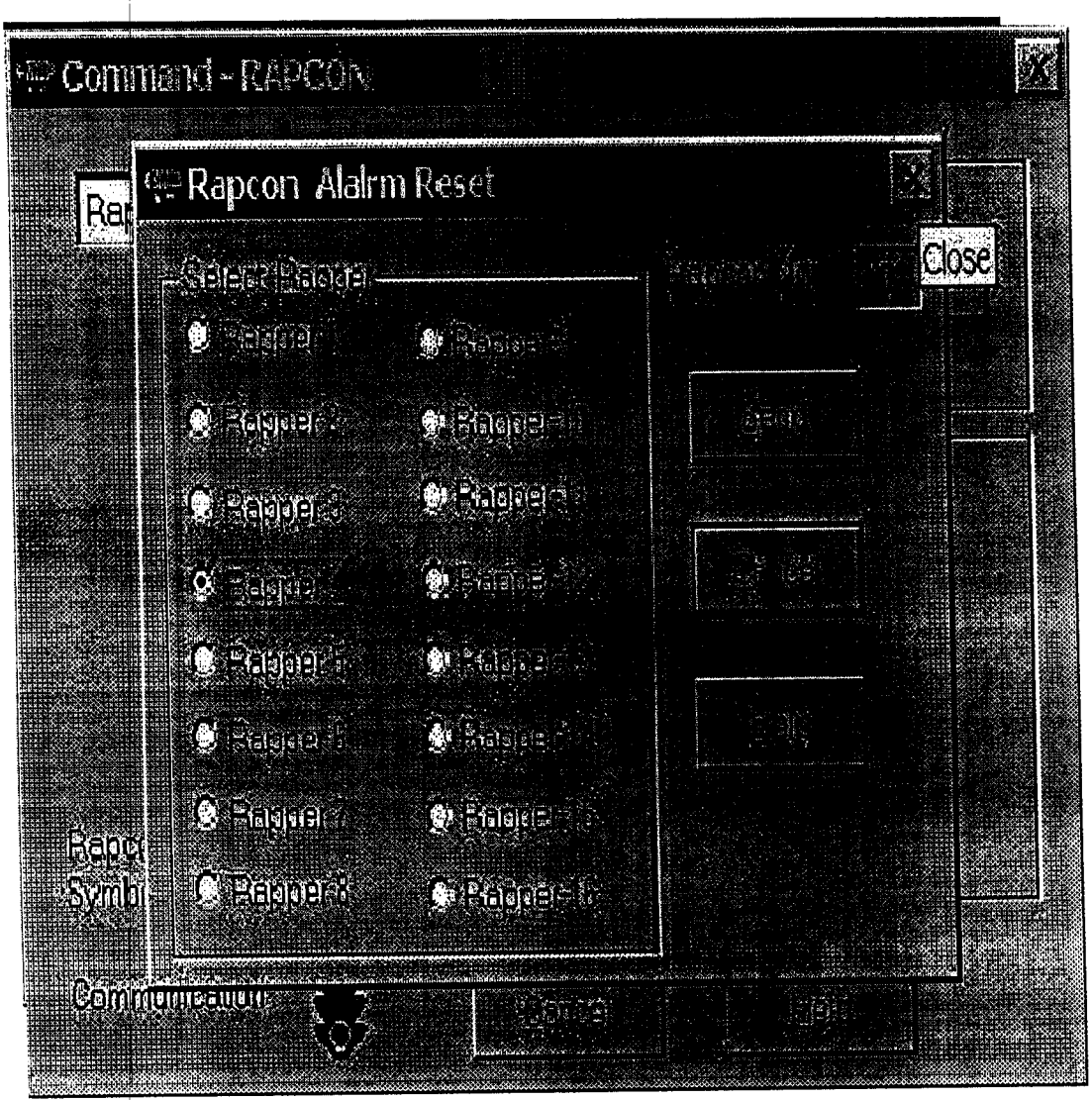

Password
Password Level
Enter Password

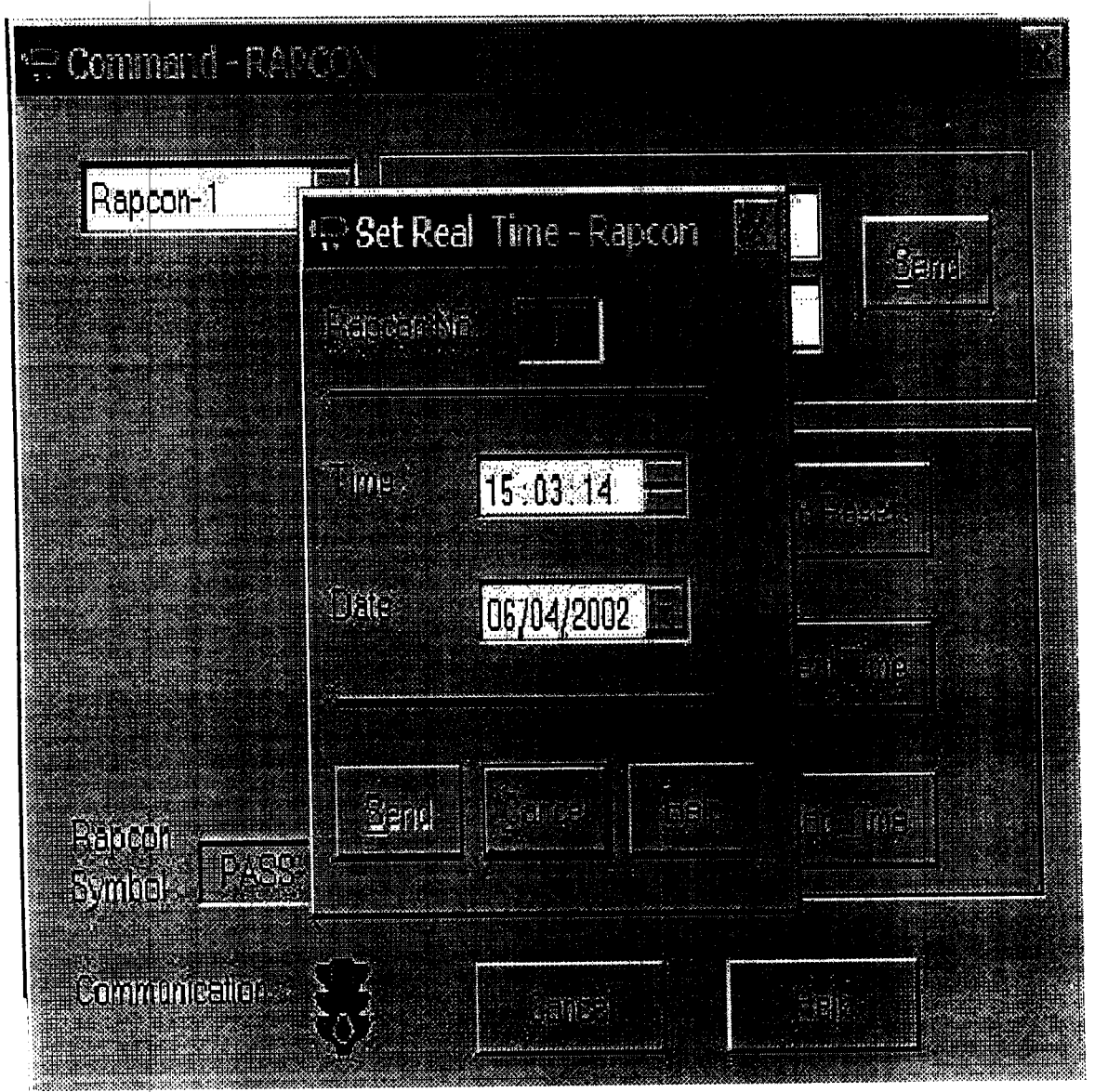


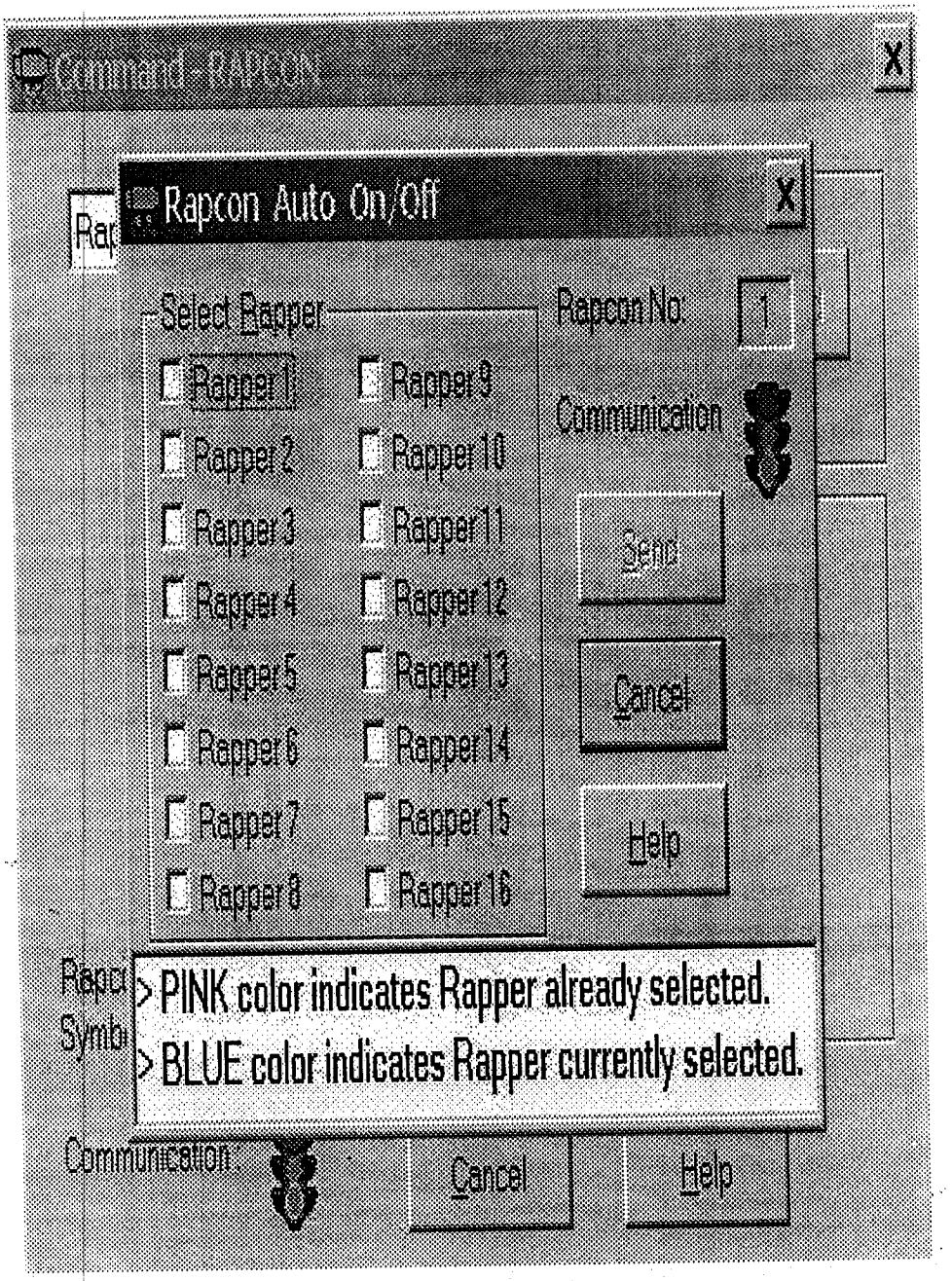
Bapcon						
T/R POWER	ON	ON	ON	ON	ON	ON
OPTIMISER MODE	ON	ON	ON	ON	ON	ON
OPT FEEDBACK	NO	NO	YES	NO	NO	NO
SPARK RATE	230	128	0	0	0	
CHARGE RATIO SET	113	5	23	3	101	
EP CURRENT(mA)	534	534	97	34	356	
EP VOLTAGE(kV)	43	54	53	60	38	
CHARGE RATIO DEP	1	1	11	31	3	
CURRENT(mA)	537	537	97	34	358	
PEAK VOLTAGE(kV)	71	89	87	99	63	
VALLEY VOLT(kV)	41	51	50	57	36	
BASE CURRENT(mA)	16	26	5	2	21	
BASE OFFSET	41	34	5	5	5	
ALARM	OFF	OFF	ON	OFF	OFF	OFF
REMOTE/LOCAL	REM	LOC	REM	LOC	REM	LOC
REPEAT FACTOR(hrs)	14	14	14	14	14	
OPTM REACHED(AUTO)	ON	ON	OFF	ON	ON	ON
FIELD SYMBOL	A1	A1	A1	A1	A1	A1

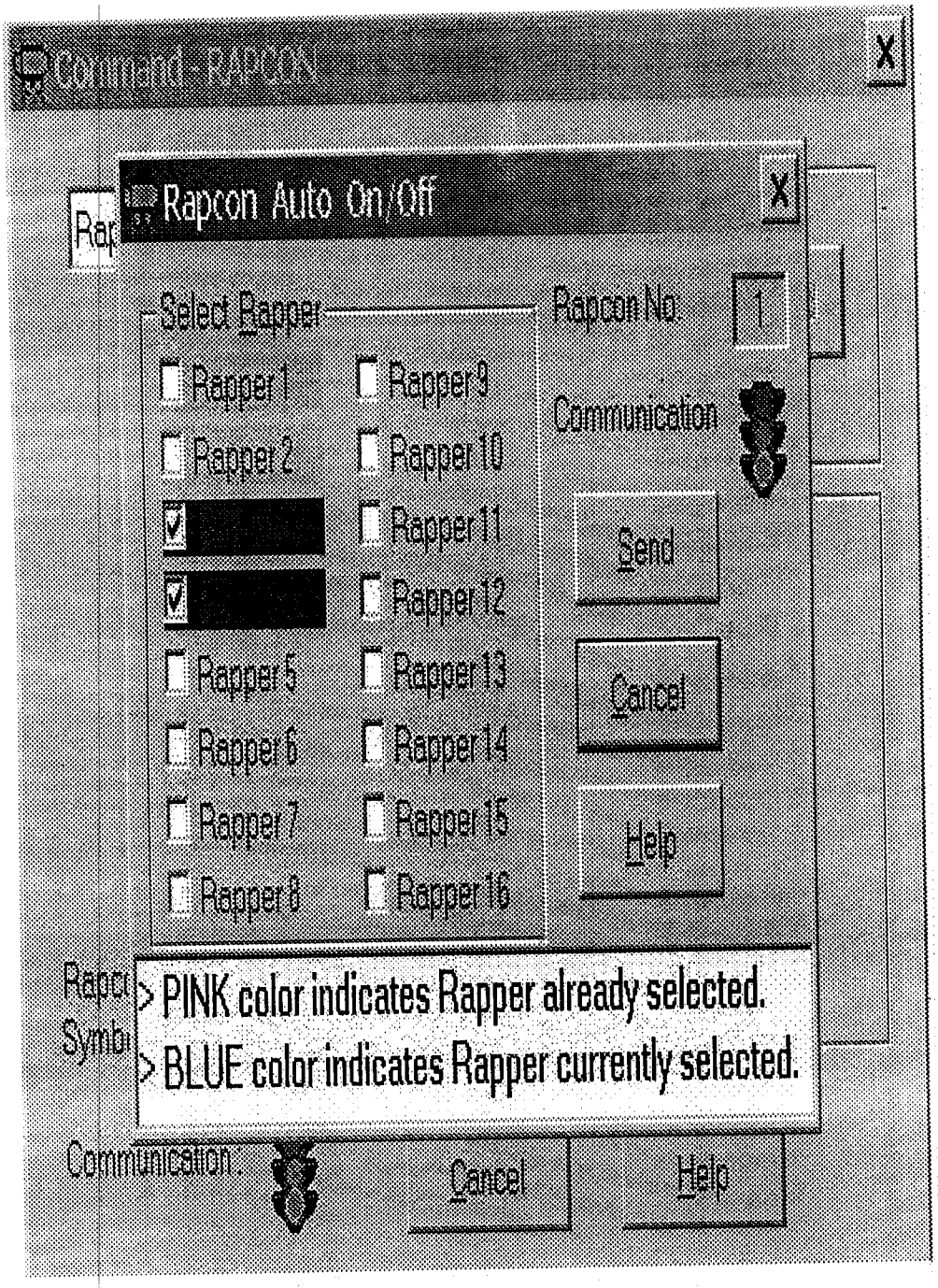

Command - BAPCON
Bapcon No
1
T/R Power
ON
None
Optimize
ON
None
Charge Rate
AV CT Limit
IC CT Limit
1222
Base Charge
Repeat Time
Alarm Status
OFF
None
Cancel
Send
Help

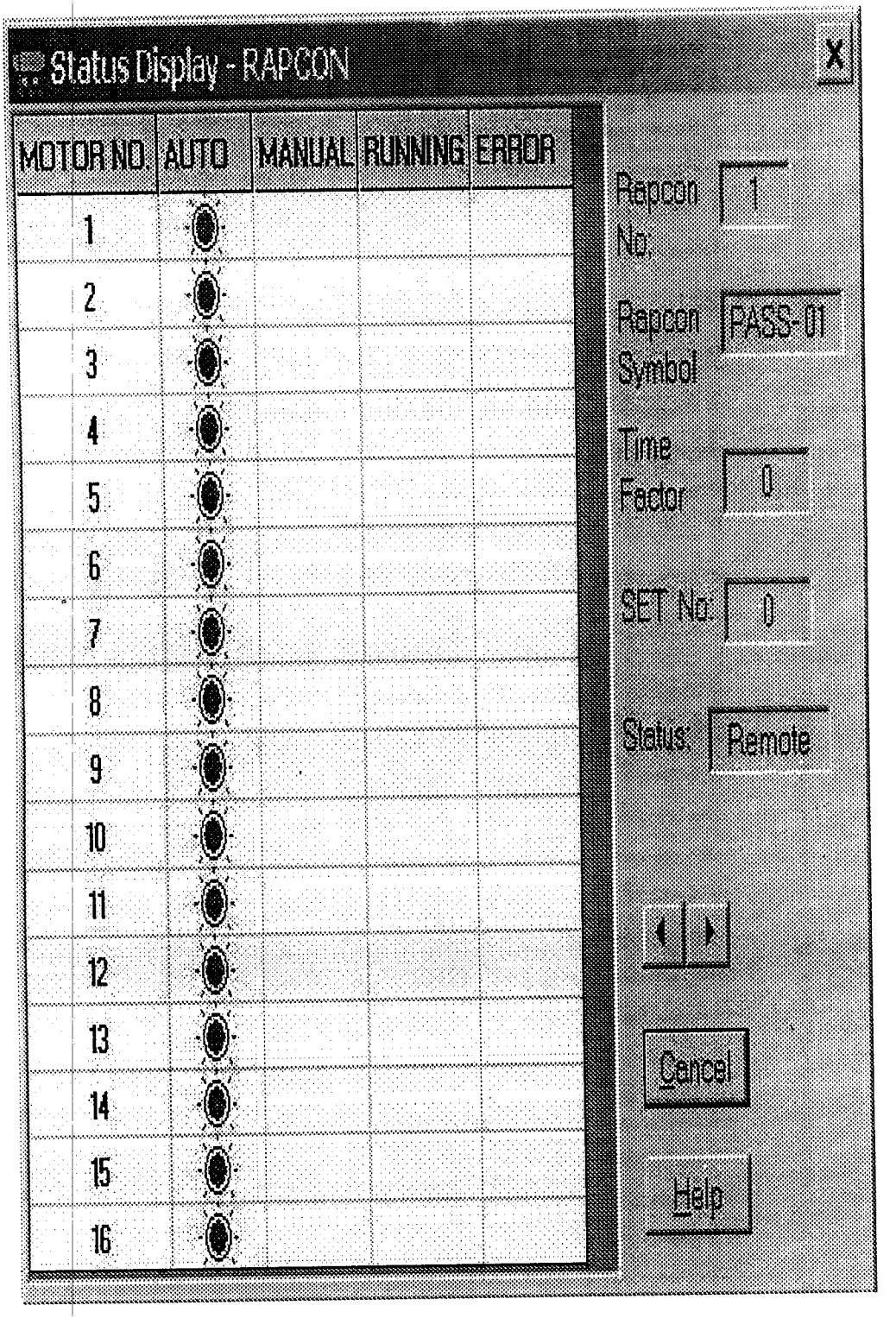

Command - RAPCON
Rapcon-1
Send
Rapcon
Symbol
PASS
Communications


Command - RAPCON
Rapcon Alarm Reset
Close


Command - RAPCON
Rapcon-1
Set Real Time - Rapcon
Send
Time:
15:03:14
Date:
06/04/2002
Send
Rapcon
Symbol:
Communication
Cancel


Command : RAPCON
Rapcon Auto On/Off
Select Rapper
Rapper 1
Rapper 2
Rapper 3
Rapper 4
Rapper 5
Rapper 6
Rapper 7
Rapper 8
Rapper 9
Rapper 10
Rapper 11
Rapper 12
Rapper 13
Rapper 14
Rapper 15
Rapper 16
Rapcon No.
1
Communication
Send
Cancel
Help
> PINK color indicates Rapper already selected.
> BLUE color indicates Rapper currently selected.
Communication
Cancel
Help


Rapcon Auto On/Off
Select Rapper
Rapper 1
Rapper 2
Rapper 5
Rapper 6
Rapper 7
Rapper 8
Rapper 9
Rapper 10
Rapper 11
Rapper 12
Rapper 13
Rapper 14
Rapper 15
Rapper 16
Rapcon No:
1
Communication
Send
Cancel
Help
> PINK color indicates Rapper already selected.
> BLUE color indicates Rapper currently selected.
Communication:
Cancel
Help


Status Display - RAPCON
MOTOR NO
AUTO
MANUAL
RUNNING
ERROR
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Repcon No:
1
Repcon Symbol
PASS-01
Time Factor
0
SET No:
0
Status:
Remote
Cancel
Help

BIBLIOGRAPHY

9. BIBLIOGRAPHY

Distributed Applications with Microsoft Visual C++.
Wilson, Scott.

Learning Visual C++ in 14 Days.
David Chapman.

Learning Visual C++ in 21 Days.
David Chapman.

Object Oriented Programming in Turbo C++.
Robert Lafore.

Digital Electronics an Introduction to Theory and Practice
William H Gothann.

MFC Internals
George Shepherd, Scott Wingo

P 727

Coimbatore LIBRARY 641 049