NAVAL POSTGRADUATE SCHOOL
Monterey, California

DESIGN AND IMPLEMENTATION OF THE PANSAT SOFTWARE GROUNDSTATION

by

Jens Bartschat

September 1995

Approved for public release; distribution is unlimited

Prepared for: Unversitaet der Bundeswehr Muenchen
85579 Neubiberg, Germany

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral M.J. Evans   R. Elster
Superintendent   Provost

This report was prepared for Unversitaet der Bundeswehr Muenchen, 85579 Neubiberg, Germany.

Reproduction of all or part of this document is authorized.

The report was prepared by:

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1995	3. REPORT TYPE Technical Report

4. TITLE AND SUBTITLE DESIGN AND IMPLEMENTATION OF THE PANSAT SOFTWARE GROUNDSTATION WITH A WINDOWS- AND WINDOWS NT BASED C++ DEVELOPMENT SYSTEM ON AN IBM-COMPATIBLE PC	5. FUNDING NUMBERS
6. AUTHOR(S) Jens Bartschat	

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER NPS-SP-95-002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Unversitaet der Bundeswehr Muenchen 85579 Neubiberg , GERMANY	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES   The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the US Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The PANSAT Software Groundstation enables a user to command and control PANSAT once it is in space, provided he has an IBM-compatible and Windows® or Windows NT® capable computer with a serial interface and the necessary additional hardware. Via the implemented PANSAT Command Language (PCL), the user will be able to access all PANSAT commands, thus control it, gather telemetry data or use its mail storing capability in an easy-to-use manner typical for Windows®-based applications.

14. SUBJECT TERMS Design and implementation of the PANSAT software groundstation on an IBM-compatible PC	16. PRICE CODE
15. NUMBER OF PAGES : 159	

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

Approved for public release; distribution is unlimited.

ACKNOWLEDGMENT

ABSTRACT

The Naval Postgraduate School's (NPS) Space Systems Academic Group (SSAG) is currently developing the Petite Amateur Navy Satellite (PANSAT). This thesis describes the PANSAT Software Groundstation development and design. This covers requirements that led to various design decisions as well as the use of the Groundstation Software and the programming background necessary to provide detailed information of how to enhance this software in the future. Furthermore, it can be considered as a brief manual for the development of software other than the Groundstation with the high-level programming language C++.

The Groundstation Software is designed to run under Windows 3.x® or Windows NT® operating systems on an IBM-compatible PC. It allows an easy visual access to all command and control features incorporated in the PANSAT design via the PANSAT Command Language (PCL). The software design provides full control over the additional hardware necessary to physically establish a connection to PANSAT, by plugging it into the serial port of the PC running the Software Groundstation.

The data structures and the visual controls are designed to meet the requirements of usability, flexibility and compatibility to third-party software. In addition, the use of data encapsulation as a typical feature of the programming language C++ ensures readable source code.

TABLE OF CONTENTS

I. INTRODUCTION

A. THE PANSAT PROJECT

The PANSAT project was initiated in 1989 as an educational program for student officers at the Naval Postgraduate School's (NPS) Space Systems Academic Group (SSAG). It was intended to prepare postgraduate students for space related tasks as well as to develop a cadre of engineers capable of developing and actually producing space qualified hardware.

PANSAT is a small satellite for digital store-and-forward communications in the amateur frequency band. It features a direct sequence spread spectrum differentially coded, binary phase shift keyed (BPSK) communication system at an operating frequency of 436.5MHz. This adds a new dimension to amateur radio communication, as spread spectrum capability has never been used before in that context. The store-and-forward capability will allow amateur radio operators to send or receive messages during several short communication windows every day, each 6 to 10 minutes of length.

The whole PANSAT structure weighs approximately 150 pounds, has a diameter of about 19 inches, and is being designed to launch as a secondary payload from the Space Shuttle as part of the Hitchhiker Program. It is made of aluminium 6061-T6 and built around a main load bearing cylinder connected to the lower equipment plate; a 26-sided polyhedron was the chosen configuration to maximize solar panel area and thus power generation. PANSAT will be unstabilized and tumble freely, once put into space with a Get Away Special (GAS) container. Its operational life is expected to be two years at an inclination between 28.5° and 51.6° and an altitude between 160 and 220 nautical miles.

B.   THE PANSAT GROUNDSTATION

Once PANSAT is in space, the only means to access it is via radio control. This shall include both commanding and gathering telemetry. Because PANSAT's hardware capabilities require commanding structure of a relatively high complexity, the problem of dealing with the correct sequence of all possible commands, their correct timing as well as telemetry storage and -maintenance becomes apparent. The most convenient way for PANSAT users would then require some kind of maintenance tool or, in terms of space technology, **groundstation** which would encapsulate this functionality in one entity.

Most commonly, satellite groundstations consist of three parts (Figure 1): one or more computers, the necessary additional hardware for radio communications, and the software serving as a connection between those two parts. The software part of such a satellite groundstation is the scope of this thesis.
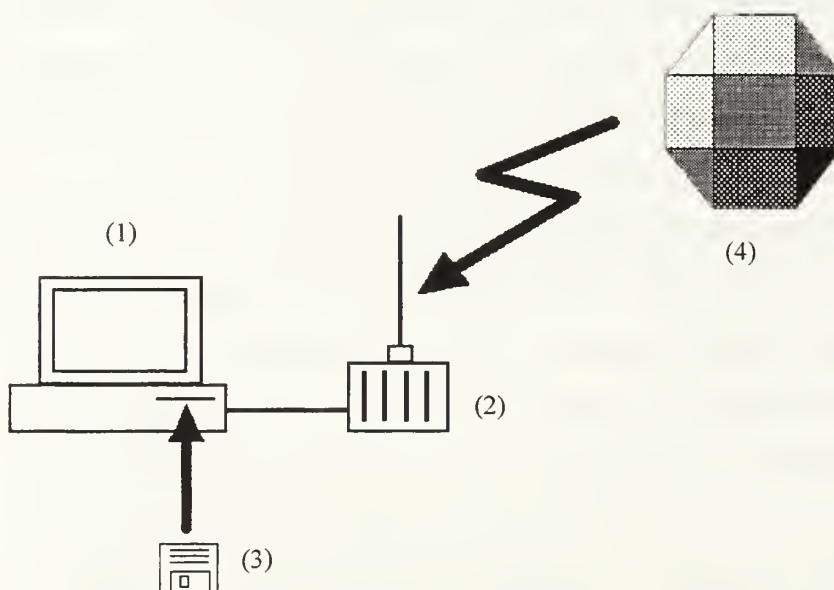
Figure 1: Parts of a Satellite Groundstation and Satellite:
A PC (1),   the   addditional   hardware   for   radio
communication (2), the Groundstation Software (3) and the
satellite to communicate to (4).

The term *Software Groundstation* will be used in the following, referring to the software part of a satellite groundstation. This is done to indicate that the software is a major and separable part of the whole groundstation concept, and to distinguish it from the hardware part.

The Software Groundstation is the only interface between users on earth and the spacecraft. As with most of all user interfaces, they should be adapted to human needs rather than human beings to user interfaces. Because many technical vehicles (such as satellites) are becoming more complex as they are becoming more powerful, the design of a satellite-to-human software interface likewise is more important. Whenever human interaction is needed for such vehicles, user friendliness should be a major concern before, while, and after the development phase of the software user interface. A user friendly design does not only make working with this interface more convenient, but also increases security and performance.

C. OVERVIEW

The following four chapters "Groundstation Requirements", "Groundstation User's Manual", "Development Preparation Manual" and "Programmer's Reference" present the reader with a climax in detailed description. The first two chapters explain necessary background information for users of the groundstation, whereas the last two chapters prepare a software developer to cope with the problem of enhancing the groundstation software.

For reference purposes, the current source code of the groundstation is provided in the Appendix.

II.   GROUNDSTATION REQUIREMENTS

This chapter provides the information which finally led to the multiple decisions concerning the development and design of a suitable PANSAT Software Groundstation. The first section explains the need for a computer based (software) groundstation. The second section discusses the more detailed aspects of the groundstation based on those decisions. Finally, the third section allows a closer inspection on the facts that led to the decisions made concerning the software development environment of the PANSAT Groundstation.

A.   THE GROUNDSTATION AS A COMPUTER SOFTWARE
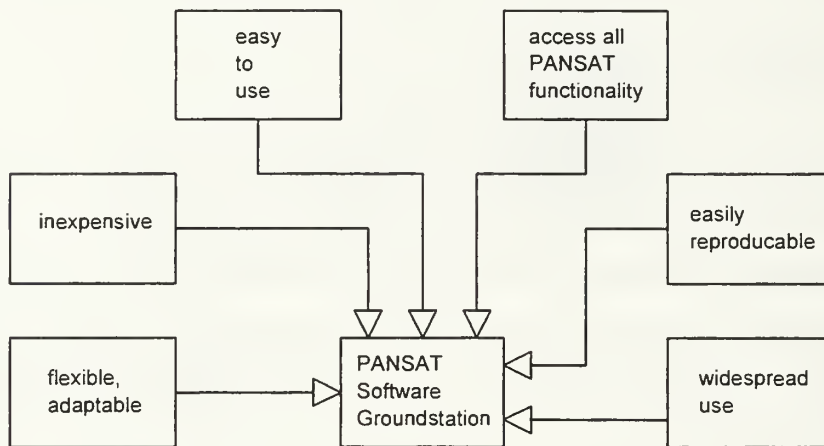
Figure 2: Advantages of a Software Groundstation

Programming a computer requires detailed knowledge of the development tools, including a programming language as well as the programming and hardware environment. The lack of this knowledge often prevents one from the use of up-to-date software tools and computers; even though a software implementation in such an

environment would be most advantageous. The advantages of a Software Groundstation instead of other non software-based solution become apparent when just enumerating the advantages of software in general:

- Software is flexible by definition: it is *designed* to be changed and adapted to a large variety of tasks. Whatever is imaginable to do with a specific hardware (computer, additional hardware) could be done by programming it.

- Once a basic knowledge about programming is established, software development can go on relatively fast and cheap. Besides a standard PC and the development package, there is hardly any need for cost-intensive additional products. However, sophisticated software tools can be very expensive, and the time spent learning to program is a cost factor too. But normally the benefits of a software implementation will be worth the money and time spent for it, and sometimes there is not an alternative.

- The flexibility of software applications gives the opportunity to program them user friendly, especially with a graphical oriented operating system such as Windows or Windows NT.

- Use of a PC as a runtime platform ensures widespread use of the software that can run on it, thus making it attractive to all people who want to solve a specific problem with this software. Also, many people own a PC.

B. SOFTWARE AND DEVELOPMENT CONSTRAINTS
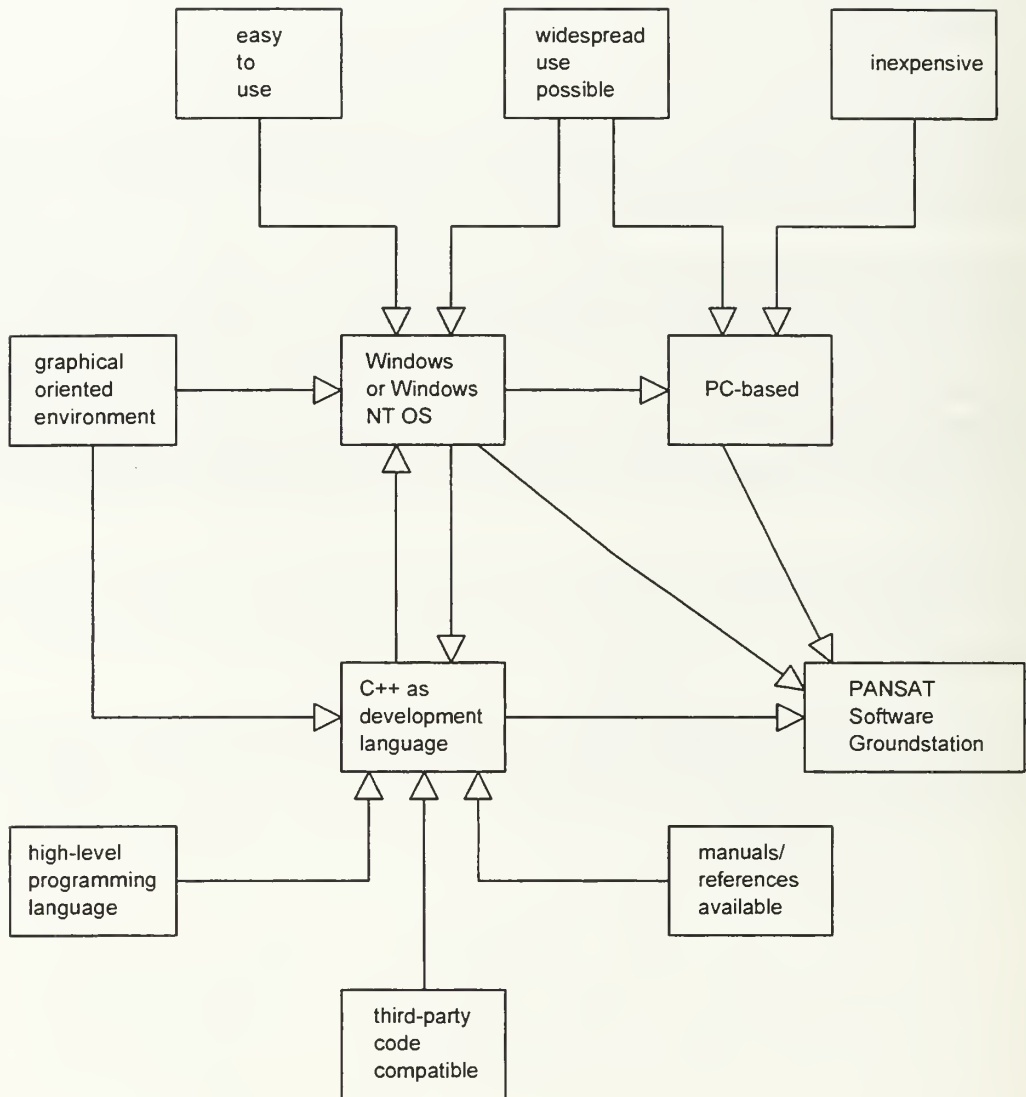
Figure 3: Software and Development Environment Constraints Leading to the Design of the PANSAT Software Groundstation

The software tools to be used when programming the PANSAT Software Groundstation were undetermined in the beginning. However, some of the requirements forced a decision:

- Existing software such as the operating system intended to be used onboard PANSAT was already available. It was written in C, so the programming language used for the Software Groundstation should be C (or C++, which incorporates C) also. However, this was not a must.

- The programming language should be high-level and structurized, and it should be common for multipurpose tasks. Furthermore, it should allow a GUI method for user interaction; that is, a huge choice of visual controls. These constraints limited the amount of suitable languages to *Object Pascal* (as used in Borland's *Delphi*) and *C++* (as used *in Microsoft Visual C++* and *Borland C++*). Delphi offered a large variety of visual controls, whereas the C++ compilers stuck to somewhat standard controls. But because of the proven performance and the long period of presence on the market, C++ was chosen.

- The operating system used for development should be the same for the runtime application version. This ensures data structure integrity and allows visual design for the application GUI. It also minimizes software incompatibility. The best and/or most commonly used operating systems enabling GUI programming are the *Macintosh Finder* and *Microsoft Windows 3.x* or *Windows NT*. Because of the common market acceptance for Windows as well as being the most common platform within the SSAG, this operating system was chosen.

- The hardware development platform should normally be the same platform as for running an application (in contrary to cross-platform development). This ensures hardware compatibility and minimizes software incompatibility between development and runtime application versions. Furthermore, a PC-based platform would be recommendable because of the widespread use of Intel®-80x86 processor series based computers.

The first steps in designing a Software Groundstation were done in accordance to [Ref. 15, chapter IX. *Ground Station Software Design*]. This description, however, involved too little detail to be useful for the actual design. In addition, the PANSAT Command Language (PCL) [Ref. 16] was subject to development after [Ref. 15] was written, and development software tools increased their capabilities during that time. So it turned out that almost none of the information in [Ref. 15] were suitable for the PANSAT Software Groundstation.

C. THE DEVELOPMENT ENVIRONMENT

For the development of the PANSAT Software Groundstation (furthermore referred to as "groundstation") several aspects have been taken into consideration. First, the groundstation should be a PC-based application using the Windows or Windows NT graphical user interface (GUI). Second, it should be written in a language familiar to the Space Systems Academic Group, so that it could be changed or enhanced once the core has been programmed. Third, this language should allow compatible access to the third-party developments, such as SCOS[1].

The choice was quite easy: the programming language should be C or C++. This decision determined the compiler: Microsoft Visual C++ Compiler (furthermore referred to as "MSVC"). There were other C++ compilers on the market, but as the author was familiar with this specific one, MSVC was chosen. The new version of MSVC runs only under Windows NT; thus, the development platform was determined.

The GUI specifications for the groundstation needed a more advanced design than MSVC was capable to offer in the first place. However, MSVC's capabilities could be enhanced by an additional interface programming package. WinWidgets/32 (WW) was chosen, especially because of its so-called "tabbed dialog" feature (a GUI feature which simplifies the display of multiple dialogs). In order to make MSVC work with WW, another dialog editor had to be purchased: Borland's Resource Workshop (RW). The MSVC built-in AppStudio dialog editor does not feature the ability to graphically manipulate a GUI in connection with WW, wheras RW offered this possibility.

---

[1] SCOS: Spacecraft Operating System; the operating system used for PANSAT

III. GROUNDSTATION USER'S MANUAL

This manual describes how the PANSAT Software Groundstation actually works. It consists of nine subsections, each representing one dialog of the groundstation which automatically appears when you start the application. Each of those dialogs represents one part of the PANSAT Command Language (PCL) with similar functionality. All eight dialogs (and a remaining embedding parent dialog) offer full access to all commands available with PCL. Finally, a short description of some auxiliary dialogs is presented.

This manual assumes that you are already familiar with using a graphical oriented operating system such as Windows or Windows NT.

A. THE "SCRIPTS" DIALOG
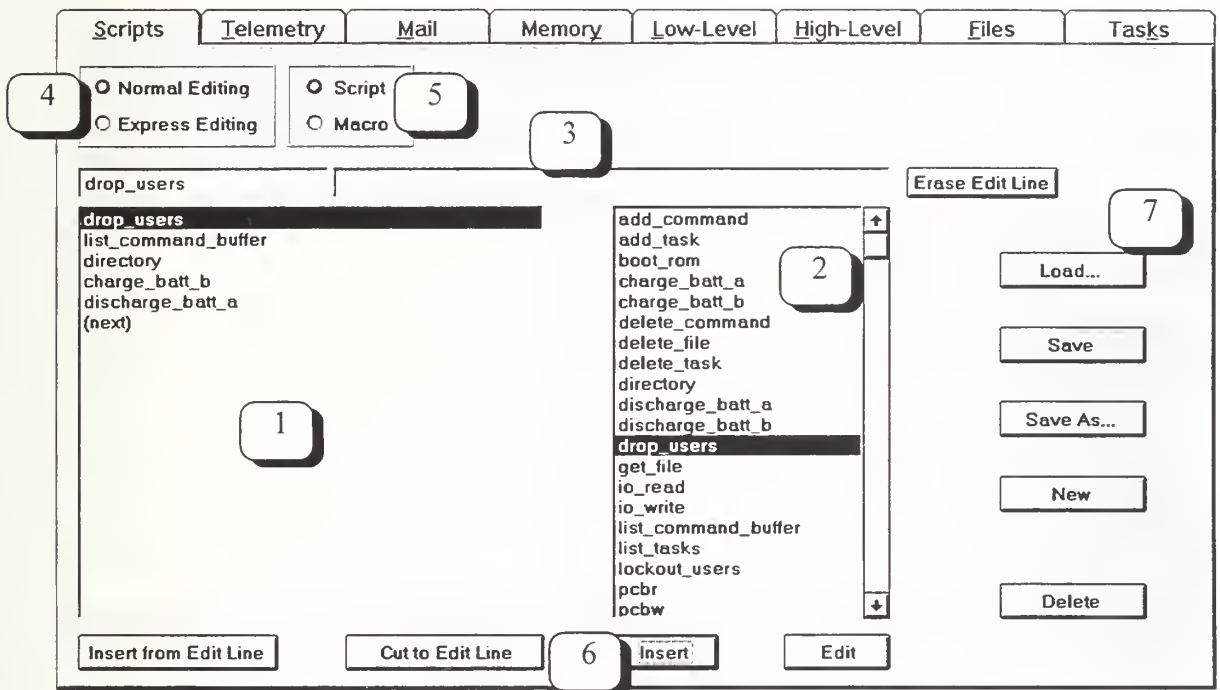
Figure 4: The Scripts Dialog

Within this `Scripts` dialog (Figure 4), the user is able to load, write, create and edit *scripts* and *macros*. Both may consist of one or more *commands*. A *command* is every order you give either to PANSAT or the Groundstation Software. A *macro* is one or more subsequent *commands* which are intended to be sent to PANSAT, that is, one or more command which is part of the PANSAT Command Language PCL. A *script* is one or more subsequent *commands*; this includes both PCL and language items not intended to be sent to PANSAT, but to control the Groundstation Software itself. Refer to Figure 4 for a description of the features of this dialog:

1. This Listbox represents the current Macro. It contains all commands in the execution sequence. All new commands are inserted before the highlighted command (`drop_users` in this case). When highlighted, the `(next)` entry allows insertion at the end of the Macro.

2. This Listbox shows all available PCL commands, or, in case of `Script` editing (5), *all* available commands.

3. These two Editfields show the command (left, `drop_users` in this case) and its parameters, if applicable (right, void in this case). The `Erase Edit Line` Button on the far right clears both Editfields. If a command takes parameters, a click in the right Editfield causes the specific dialog to appear in which the parameters of the command (left Editfield) can be edited. This will be one of the following dialogs: `Mail`, `Memory`, `Low-Level`, `High-Level`, `Files` or `Task` dialog. They are put into a special mode which allows only for setting or editing parameters of the command in the left Editfield.

4. These two Radiobuttons determine the edit mode. They change the reaction on mouse clicks in area 1 and 2. The four buttons in area 6 indicate which mouse click produces which reaction, that is, left or right mousebutton. Refer to area 6 for explanation.

5. These two Radiobuttons determine whether *all* commands are to be shown in areas 1, 2 and 3 (`Script`) , or *only PCL commands* are to be shown (`Macro`).

6. These four Pushbuttons allow Macro editing. `Insert from Edit Line` inserts the contents of 3 before the highlighted command listed in 1. `Cut to Edit Line` deletes the highlighted command line in 1 and pastes it into 3. `Insert` pastes the highlighted command of 2 in both 1 and 3, but only in case the command does not require parameters. However, if it requires parameters, the highlighted command in 2

is pasted only in 3. By pressing `Edit`, the highlighted command in 2 is pasted only to 3, regardless of its parameter requirements. In `Express Editing` mode, the position of the Pushbuttons below the Listboxes 1 and 2 indicates which mousebutton (left or right) has to be clicked in 1 or 2 to obtain the same reaction as when clicking on one of the Pushbuttons. For example, click left in 1 to `Insert from Edit Line`, click right in 1 to `Cut to Edit Line`. Same with 2: click left in 2 to `Insert`, or click right in 2 to `Edit`.

7. These five Pushbuttons allow loading, saving and creating Macros as well as deleting Macros from disk. If a Macro has been changed, the user is prompted to *save* or *save as* it (depending on whether or whether not it already has a filename) before continuing with other actions. Furthermore, the default directory setting is automatically updated according to the Radiobutton setting 5. Scripts and Macros are stored in different directories as defined in the `Preferences` dialog (Figure 13), or as the appropriate section in the `GND.INI` file tells.

B.   THE "TELEMETRY" DIALOG

This dialog presents the current telemetry data received from PANSAT. It is not yet defined whether all available telemetry should be shown, or just parts of it. Furthermore, the storage format is not implemented yet; all telemetry shall be stored in an ODBC-compatible format. This implementation might alter the decision which part of the telemetry shall be presented in this dialog. In addition, the return structure (the `SReturnCmd` structure) is not implemented yet either. This task should be accomplished after the evaluation of the input structures. Therefore, the outward appearance of this dialog is postponed after the necessary preliminary actions are taken.

The Preferences dialog (Figure 13) contains information about the location of telemetry data on a groundstation mass storage device. Telemetry data will be stored in this directory.

C.   THE "MAIL" DIALOG
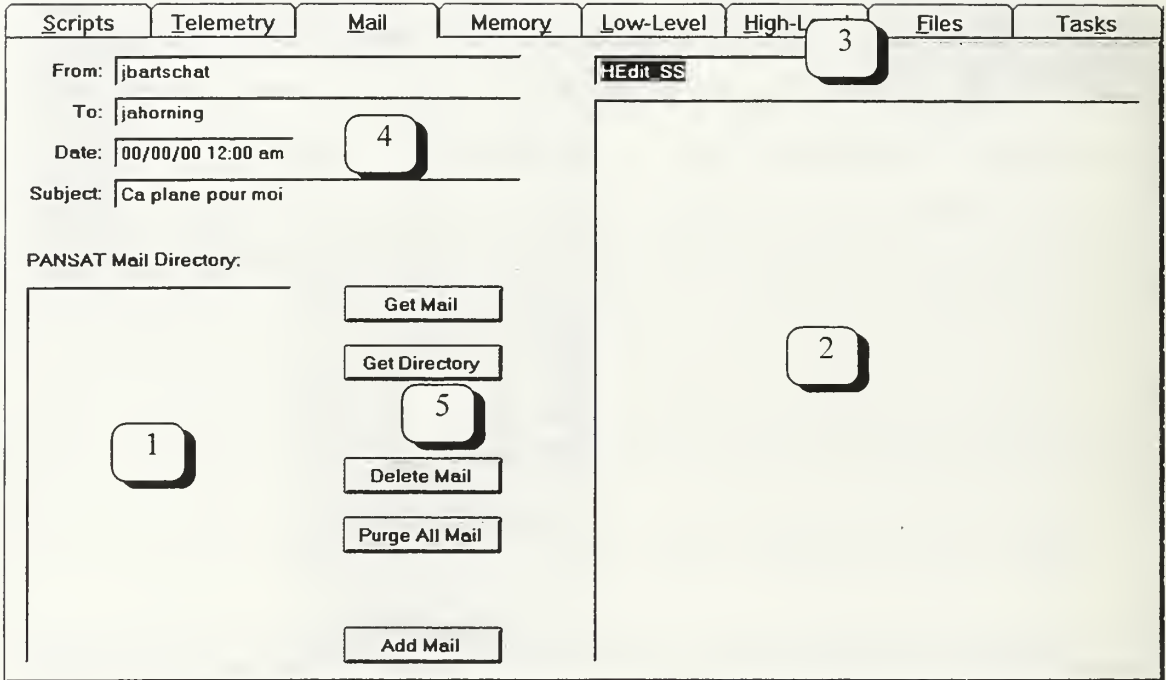
Figure 5: The Mail Dialog

The `Mail` dialog handles the PANSAT *mail* capability. *Mail* could be regarded as a message with additional information: a *from* and *to* designator, the *time* the mail was sent, a *subject* and the *text* of the ASCII-message to be included in the mail. This dialog now offers the possibility to load, create and edit a message, and send it to PANSAT; to retrieve an already stored mail from PANSAT's memory, read the message and the additional information, and save it to disk. The following visual controls allow mail handling (refer to Figure 5):

1. This Listbox shows the current status of the PANSAT Mail Directory, as stored in recent telemetry. It contains the names of the mail files. Select a mail by highlighting its name with a mouse click.

2. This Listbox shows the ASCII message of the mail. Carriage return is inserted automatically.

3. This Editfield shows the file name of the current mail.

4. These four Editfields contain additional mail information, very similar to email. The **From** and **To** designator determine the sender and the recipient of the mail, the **Date** determines the time (day/month/year hour:minute) of sending, and the **Subject** line should give an idea of what to expect from the message (as shown in 2).

5. These five Pushbuttons trigger the PANSAT mail input/output. **Get Mail** retrieves the mail currently highlighted in 1 and shows it by filling up 2, 3 and 4. **Get Directory** updates the entries in 1 by retrieving the current mail directory from PANSAT. **Delete Mail** deletes the current selection in 1 by deleting the mail inside PANSAT, and **Purge All Mail** deletes all mail inside PANSAT. A **Get Directory** afterwards should reflect these actions. The **Add Mail** button lets the user edit his own mail in 2, 3 and 4. The button then changes to **Send Mail**. When done, the user presses the **Send Mail** button, and the mail is uplinked to PANSAT.

D. THE "MEMORY" DIALOG
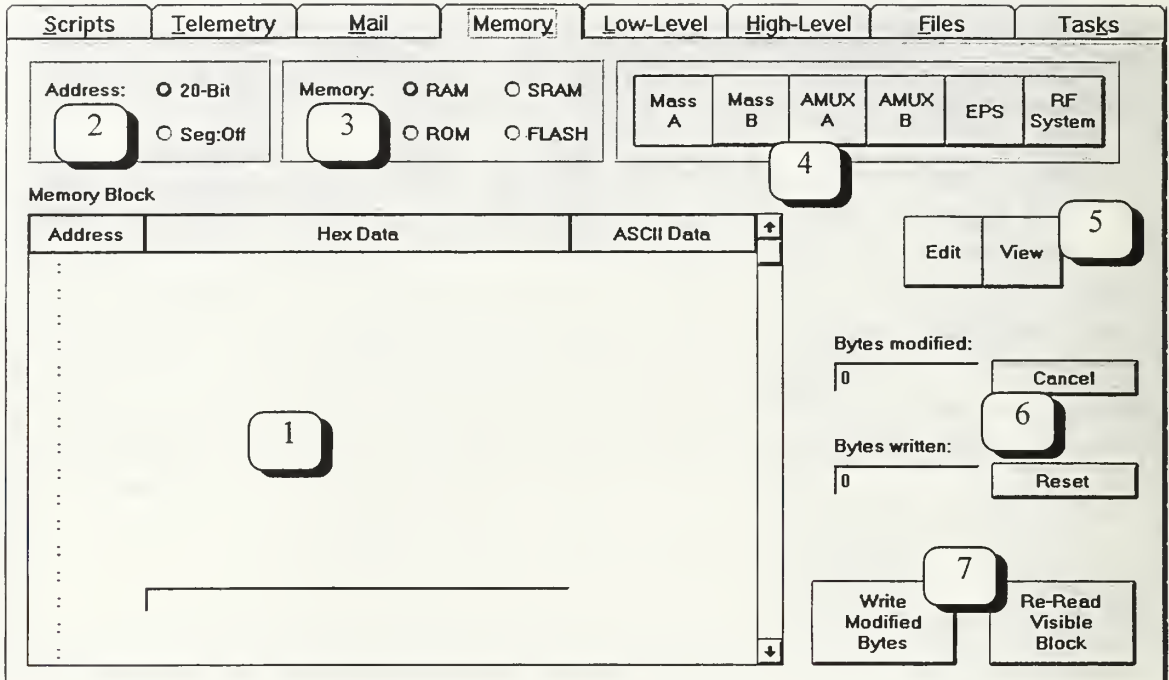
Figure 6: The Memory Dialog

The `Memory` dialog allows access to the various kinds of memories inside PANSAT. This is a very powerful feature, because every byte of PANSAT's memory can be altered. The dialog is intended to serve as an emergency repair possibility only. By using the features of this dialog, the user must be aware of the fact that he could endanger the electronic life of PANSAT.

The dialog consists of several parts (refer to Figure 6):

1. This Grid contains 256 bytes of memory data in 16 rows. The first column `Address` shows the address of the memory contents as used in PANSAT. The second column `Hex Data` presents the memory data in hex format, 16 bytes per row, separated by whitespaces. The third column `ASCII Data` shows the same 16 bytes as in `Hex Data` in ASCII format. The user may edit entries in `Hex Data` and `ASCII Data` columns. However, altered data changes its color from black to red, but is not yet uplinked to PANSAT. The Scrollbar on the right side allows sequential reading of PANSAT's memory contents; one click on the up or down arrow reads 16

bytes, one click above or below the slider reads 256 bytes. Dragging the slider with the mouse allows placing it to the desired memory range. In this case, the **Address** column contents change as the slider is dragged and shows the current 256-byte address block, but a 256 byte memory block is read from PANSAT only when the slider is released from dragging.

2. These two Radiobuttons determine the display mode of the **Address** column contents in Grid 1: either 20-Bit or Segment:Offset address display.

3. These four Radiobuttons determine the memory which is shown in Grid 1: RAM, SRAM, ROM or FLASH memory. ROM cannot be edited.

4. With these six Pushbuttons, the user can define which part of memory is to be accessed: memory in Mass Storage A or B, Analog Multiplexer A or B, the electric power supply (EPS) or the communications unit (RF System). In Figure 6, the **Mass A** (Mass Storage A) button is selected.

5. With these two Pushbuttons, the user can determine whether to allow memory editing (**Edit**) or deny it (**View**). When denied (**View**), editing in Grid 1 becomes impossible.

6. These two Editfields and two Pushbuttons reflect the current memory edit status of the memory block subject to editing[2]. In the **Bytes modified** Editfield the amount of edited (red), but not yet uplinked data bytes is shown. The **Bytes written** Editfield shows the amount of edited bytes already written to PANSAT. The **Cancel** button abandons all changes to the memory block subject to editing and reverts its data bytes to the previous values. Thus, the **Bytes modified** entry becomes zero, and all red data in Grid 1 changes its value to the previous setting and becomes black. The **Reset** button sets the **Bytes written** entry to zero, but does not do anything else to memory data.

7. These two buttons are the only ones which establish PANSAT I/O besides the Scrollbar in Grid 1. The **Write Modified Bytes** button writes all edited bytes shown red in Grid 1 as well as all edited bytes not in the current display of Grid 1 to the appropriate address and storage (as depicted in 1, 3 and 4) onboard PANSAT. The **Re-Read Visible Block** button reads the 256 bytes currently visible in Grid 1 from PANSAT and displays it. Any changes to the visible part of memory are abandoned by this action; thus, every edited (red) entry is updated with the current

---

[2] The memory data display is limited to 256 bytes in Grid 1, but the editable range can be much bigger (using the Scrollbar in Grid 1). The memory edit status as referred to in 6 includes all edited bytes (even if they are not visible in Grid 1) since the beginning of the edit session or the last uplink to PANSAT.

PANSAT memory contents and is shown in black, and the `Bytes modified` entry (6) is adjusted appropriately. This function is particularly useful when monitoring frequently changing memory contents.

E.   THE "LOW-LEVEL" DIALOG
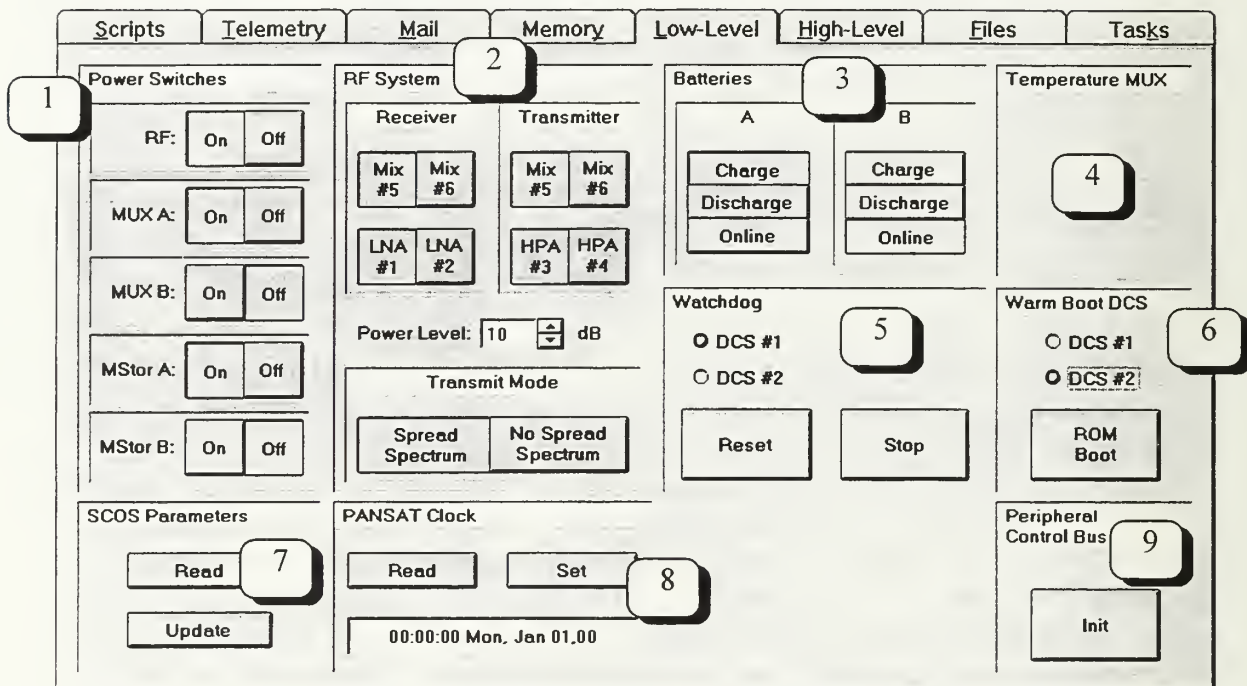
Figure 7: The Low-Level Dialog

The `Low-Level` dialog handles a multitude of settings onboard PANSAT which relate to close-to-hardware components control. The dialog is designed to provide a system overlook; thus it contains more visual controls than other dialogs. It falls into several parts (Figure 7):

1. The `Power Switches` frame contains `On-Off` switches for multiple hardware sections onboard PANSAT: the communications unit (`RF`), the multiplexing units A and B (`MUX A, MUX B`), and the mass storage devices A and B (`MStor A, MStor B`). `On` means, the unit is provided with current, `Off` means, no current available for the unit.

2. The **RF System** frame falls into four parts: **Receiver**, **Transmitter**, **Power Level** and **Transmit Mode**. The **Receiver** and **Transmitter** Mixer settings are interdependent: when using **Mix #5** for one of them, it is desirable that the other also uses **Mix #5**, and the same with **Mix #6**. However, it might turn out to be necessary to have different Mixer switches for receiving and transmitting. The first choice would be to use **Mix #5** and **LNA #1** for the receiver, and **Mix #5** and **HPA #3** for the transmitter. The receiver may use either low noise amplifier (**LNA #1** or **LNA #2**), and the transmitter may use either high power amplifier (**HPA #3** or **HPA #4**). The numbering refers to bits in the control byte determining the receiver and transmitter settings. The **Power Level** Spin Control allows for settings in the range from 0 to 255 dB; however, the exact attenuation level is not determined yet. The **Transmit Mode** subframe allows for **Spread Spectrum** or **No Spread Spectrum**; settings for Narrow Band transmission and Binary Phased Shift Keying (BPSK) are set automatically.

3. The **Batteries** frame contains settings for both batteries, A and B. The possible settings are **Charge**, **Discharge** and **Online**. During system start, it is possible (though not desirable) to have the batteries in none of those three modes. There are several combinations which are not possible: for each battery, it is prohibited to to **Charge** and be **Online**, or **Discharge** and be **Online**, or to **Charge** and **Discharge**, or to **Charge** and **Discharge** and be **Online**. Furthermore, it is not allowed that both batteries **Charge** at the same time, or that one battery **Charge**s the same time the other battery is **Online**.

4. The **Temperature MUX** (Multiplexer) switch is not determined yet.

5. The **Watchdog** Radiobuttons and Pushbuttons control the Digital Control System (**DCS #1** or **DCS #2**). Each chosen DCS watchdog can be halted by pressing **Stop**, and it can be reset by pressing **Reset**.

6. Each Digital Control System's ROM can be rebooted by pressing the **ROM Boot** button.

7. The **SCOS Parameters** frame allows for **Read** and **Update** Spacecraft Operating System's Parameters. It is not yet defined how data transfer should be accomplished for this task.

8. The PANSAT Clock frame contains an Editfield showing the current PANSAT system clock. In case it differs from ground time, the user may click the **Set** button to adjust PANSAT's system clock to the Software Groundstation time. This should only be done directly after reading PANSAT's system time via the **Read** button, because the time display is not updated automatically.

9. The Peripheral Control Bus (PCB) onboard PANSAT can be initialized inside the `Peripheral Control Bus` frame. This will reset it to a defined startup status.

F.   THE "HIGH-LEVEL" DIALOG
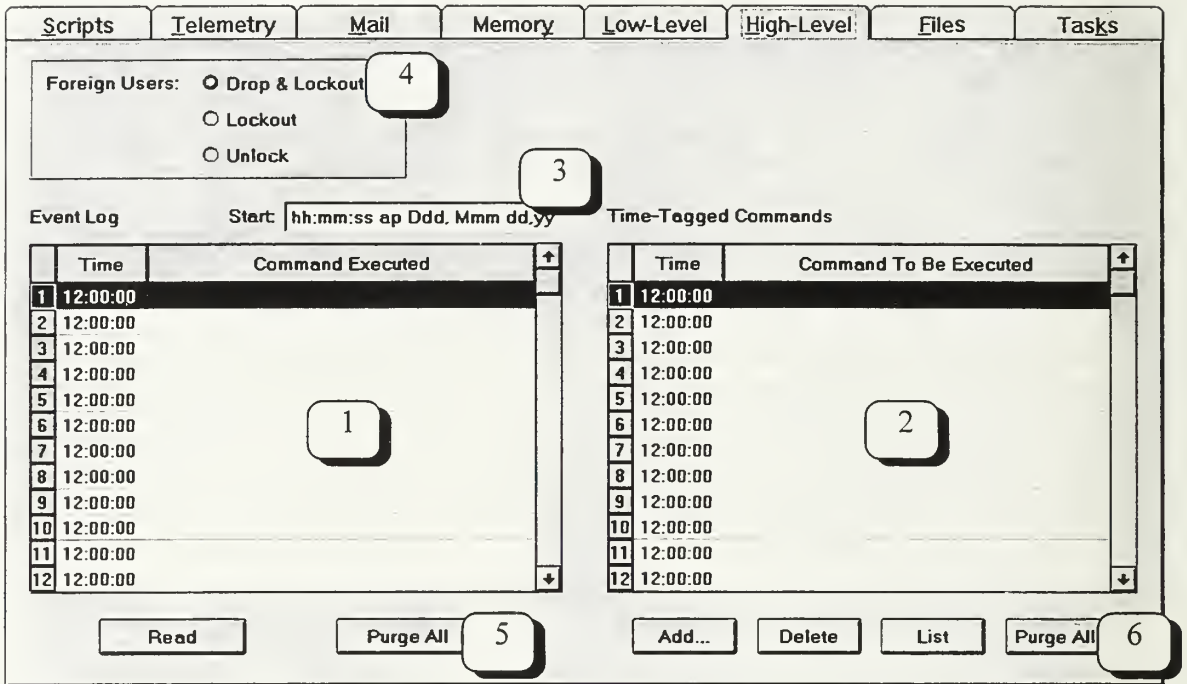
Figure 8: The High-Level Dialog

The Software Groundstation's `High-Level` dialog (Figure 8) handles PANSAT's *Event Log* and *Time Tagged Commands* feature. An Event Log is a list of time associated with an already executed command; Time Tagged Commanding allows remote command execution by associating a command to a certain time at which the command shall be executed. Thus, both lists contain the same time-to-command association. The Event Log is a report of what already happened, whereas the Time Tagged Command list holds commands for future execution. The following visual controls allow Event Log and Time Tagged Command handling:

1. This Grid contains PANSAT's current Event Log. The list may display up to 12 time-to-command associations. The `Time` column contains the time in hours:minutes:seconds of PANSAT time and shows at which time the command depicted in the same row under the `Command Executed` column has been executed.

2. This Grid holds all currently activated Time Tagged Commands. It is built up similar to Grid 1.

3. The `Start` Editfield contains the exact time and date the current Event Log has started. This time is given in hours:minutes:seconds day, month, year format.

4. These three Radiobuttons determine what status PANSAT is set to as far as user access is concerned. `Drop&Lockout` executes the appropriate PCL commands and ceases eventual foreign user access and prevents new foreign users from logging into PANSAT. `Lockout` just prevents foreign users from logging on, and `Unlock` finally allows them to again log into PANSAT.

5. The two Pushbuttons below Grid 1 refer to PANSAT's Event Log. `Read` retrieves the current Event Log. `Purge All` deletes it from PANSAT's memory, clears all Grid 1 entries and resets the `Start` Editfield.

6. These four Pushbuttons below Grid 2 handle Time Tagged Commanding (TTC). The `Add...` button adds a PCL command to the list. It invokes the `Scripts` dialog and puts it into a special mode so that only *one* PCL command (and its parameters, if applicable) may be selected. Then the command is pasted into Grid 2, where the user has to define the time he wishes the command to be executed. This time-to-command association then is sorted into Grid 2 (by time criterion). The `Delete` button erases the highlighted time-to-command association from PANSAT's TTC list. The List button retrieves the currently activated list of PANSAT's TTC. The `Purge All` button finally erases all currently activated entries from PANSAT's TTC list.

G.   THE "FILES" DIALOG
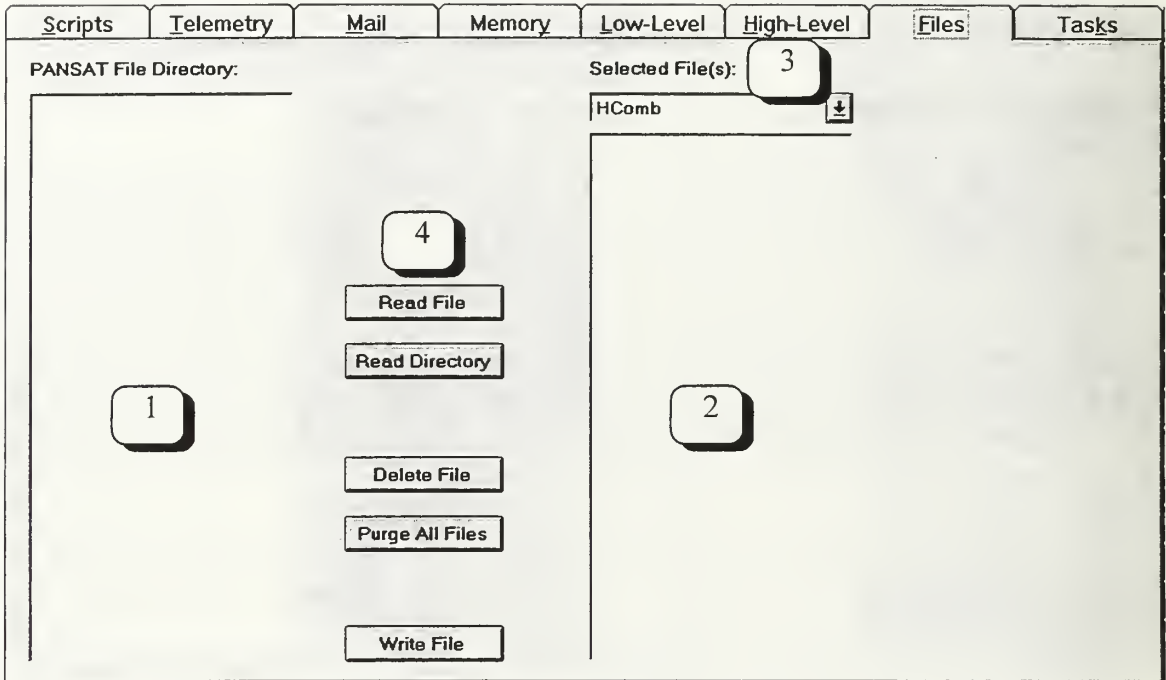
Figure 9: The Files Dialog

The **Files** dialog enables the user to access all file related functionality via PCL. A *file* is every portion of data associated with a filename by SCOS, and stored in SCOS-accessible memory. Mail, for example, is stored as a file. The following **Files** dialog items shall be implemented:

1. This Listbox contains the current PANSAT filename list as known in recent telemetry.

2. In this Listbox, the contents of the file named as shown in Editfield 3 is displayed.

3. This Combobox under the **Selected File(s):** Static Display contains a list of all recently read files. These files may be stored in the location identified by the **IN** section of the **Preferences** dialog (Figure 13), or the appropriate entry in the **GND.INI** file.

4. These five buttons enable file reading, deleting and writing from and to PANSAT. **Read File** reads the highlighted file as depicted in 1 and displays its contents and

name in 2 and 3. **Read Directory** retrieves PANSAT's current file directory and places this list in 1. **Delete File** erases the highlighted filename entry from PANSAT's file directory, and Purge All Files erases the entire filename list from PANSAT's file directory. **Write File** finally writes a file to PANSAT's mass storage device and creates an entry in PANSAT's file directory. This file shall be loaded from a groundstation storage device to 2, from where it could be uplinked via **Write File**. The appropriate buttons for loading a file from a groundstation storage device are not implemented yet. The files intended to uplink to PANSAT could be stored in the location identified by the **OUT** section of the **Preferences** dialog (Figure 13), or the appropriate entry in the **GND.INI** file.

H. THE "TASK" DIALOG
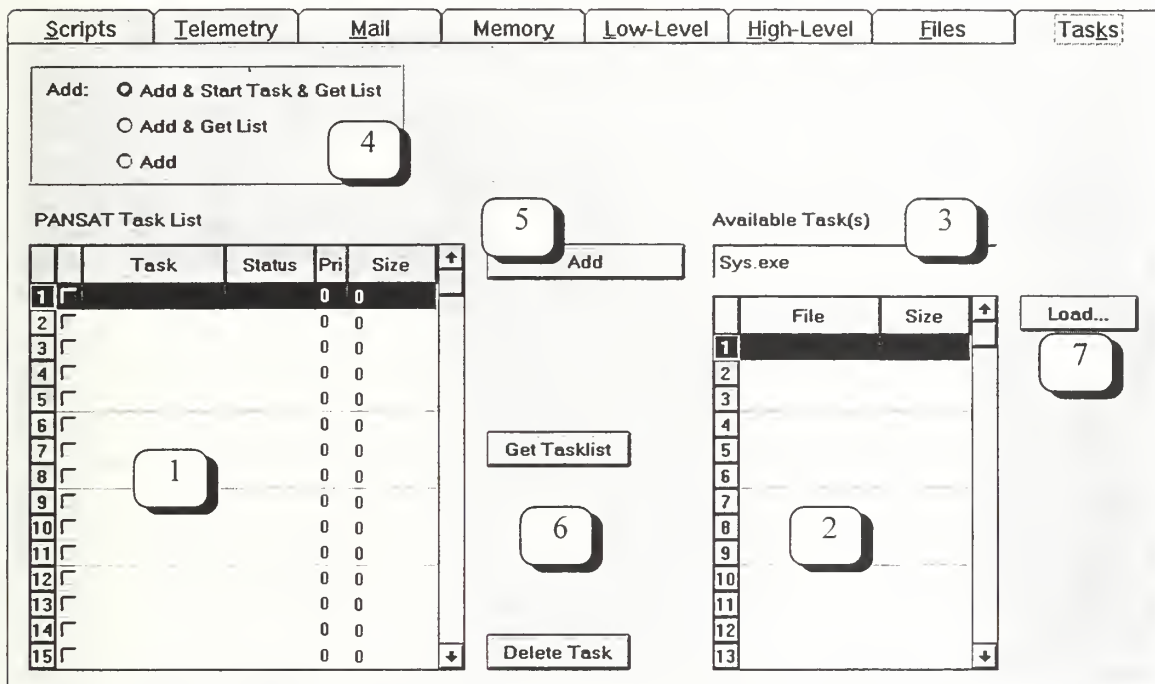
Figure 10: The Tasks Dialog

The **Tasks** dialog as depicted in Figure 10 enables the user to perform task handling. A *task* is a PANSAT executable which is added to the SCOS-maintained task list. With SCOS, a *Priority-based Round Robin* task scheduling method is used. The following visual control reflect this functionality:

1. This Grid contains task information for tasks currently recognized by SCOS. Five colums provide the necessary task information. The first column contains Checkboxes showing an activated (crossed) or deactivated (not crossed) status. The second column `Task` displays the task name; the third, `Status`, the current task status: running, waiting, and screech. Refer to [Ref. 17] for further information. The fourth column `Pri` shows the task priority in four possible levels: 0 is highest, 3 is lowest ([Ref. 17, page 11]). The fifth and last column, `Size`, depicts the length of the task in bytes. It may be as long as 2^{16} -1= 65535 bytes. Only the first column (activation status) and the `Pri` column allow for user editing.

2. This Grid contains all files stored in the location referenced by the `Task List` section of the `Preferences` dialog (Figure 13). The Software Groundstation considers all files in this directory as valid PANSAT executables. Filename and file size are shown in the appropriate columns.

3. This Editfield contains the name of the highlighted task from Grid 2.

4. These three Radiobuttons determine what action is executed when pressin the `Add` button 5. `Add & Start Task & Get List` lets the `Add` button perform all these three actions; the `Add & Get List` and `Add` Radiobuttons let the `Add` button perform actions respectively.

5. The `Add` Pushbutton uplinks the task with the taskname as shown in 3 to PANSAT's Task List. Depending on the settings in 4, this button behaves different. When `Add & Start Task & Get List` is activated, a click on the `Add` button is the same as adding a new task to PANSAT's task list, clicking its activation checkbox in 1 and pressing the `Get Tasklist` button. When `Add & Get List` is activated, a click on the `Add` button is the same as adding a new task to PANSAT's task list and pressing the `Get Tasklist` button. If only the `Add` radiobutton is activated, a click on the `Add` button is just the same as adding a new task to PANSAT's task list.

6. The `Get Tasklist` button retrieves the current task list from PANSAT, and the `Delete Task` button erases the task currently highlighted in 1 from PANSAT's task list.

7. The `Load...` button allows loading PANSAT executables from other locations than shown in the `Preferences` dialog in the `Task List` Editbox by bringing up an appropriate Common Dialog Box.

I. THE EMBEDDING MACRO/STATUS DIALOG

All dialogs explained above are embedded in a main dialog template which extends the Software Groundstation functionality by a *Macro List* (Figure 11) and a *Status Display* (Figure 12).

The Macro List at the far right side of the main dialog contains the following visual controls:

1. Up to 15 Macros may be accessed by the pushbuttons **0** to **15**. By clicking these buttons with the left mousebutton, the macro stored in this place may be executed. Macro language allows recursive execution of macros (Macro A invokes Macro B, and Macro B invokes Macro A, ...). By clicking one of these buttons with the right mousebutton, the **Macro** dialog appears (to be determined). From within this dialog, the user may load a new macro, and give it a name other than just a number, which then will be displayed instead of the number as shown in Figure 11. If not every macro button is occupied, the macro list is limited to the current amount of macros.
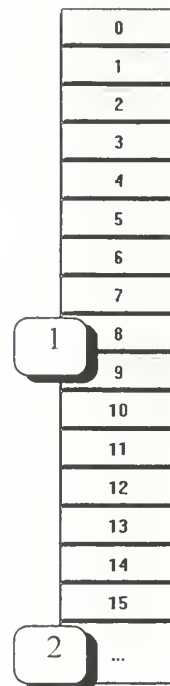
Figure 11: The Macro List

2. The . . . button switches to the following selection of 15 macros from the *macro list*, or, if the end of the list is reached, to the first 15 macros. The macro list can be much longer than just 15 macros. This button allows for switching in portions of 15 macros.

The Status Display on the bottom of the main dialog contains the following items:

1. The **Send** and **Receive** color frame. When sending (that is, transmitting data through the serial port), the dark red color turns to bright red. When receiving (obtaining data from the serial port), the dark blue color turns to bright blue.
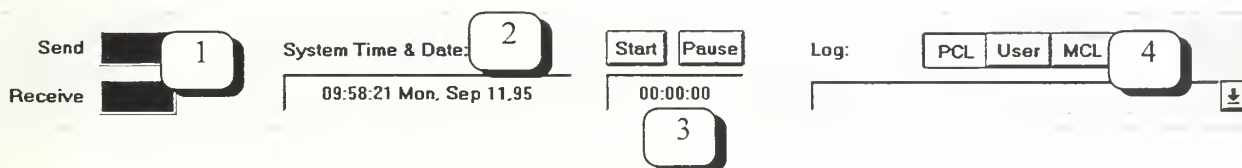
Figure 12: The Status Display

2. The `System Time & Date` Editfield reflects the current Software Groundstation internal time and date.

3. This short period timer can be started by pressing `Start` or halted by pressing `Pause`. When pressing `Start` while pausing, the timer is reset to 00:00:00. Pressing `Pause` when pausing resumes the timer. This timer may become particularly important in order to achieve a time perception of how long PANSAT will be above the earth horizon. It could be started when first establishing contact with the satellite and thus measure the communication period.

4. The `Log` Combobox reflects all actions done with the Software groundstation. It contains *Event Log*-like entries which then are stored under a username-specific filename according to the login (refer to Figure 14) in the directory referenced to by the `User Log` Editlist entry of the `Preferences` dialog (Figure 13). Three Pushbuttons determine what type of user action is shown in the `Log` Combobox: `PCL` means that only PCL commands are shown in the Log; `MCL` means, *all* commands (including PCL commands **and** the Software Groundstation commands as used for the *Script* language) are shown in the Log. `User` then advises the Log only to reflect all Script language commands *without* the PCL commands.

J.   THE LOGIN AND PREFERENCES DIALOG
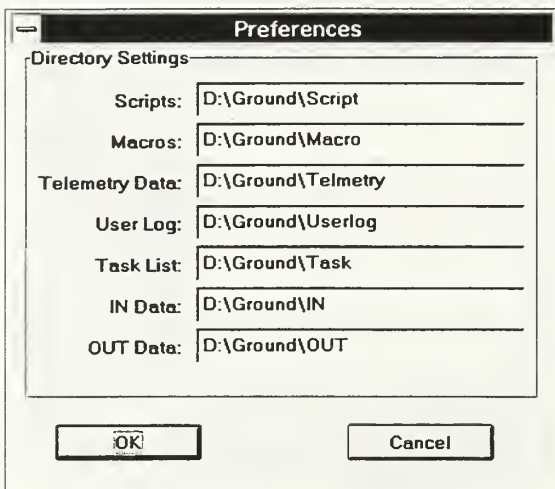
Figure 13: The Preferences Dialog

The `Preferences` dialog as shown in Figure 13 reflects the contents of the `GND.INI` file. Every time it is invoked by the appropriate menu item, the `GND.INI` file is read. When clicking `OK`, the current contents of the dialog is saved to `GND.INI`. `Cancel` abandons eventual changes.

The Editfields of this dialog determine the storage directory for various groundstation settings. Refer to the description of the appropriate dialog above for further details.
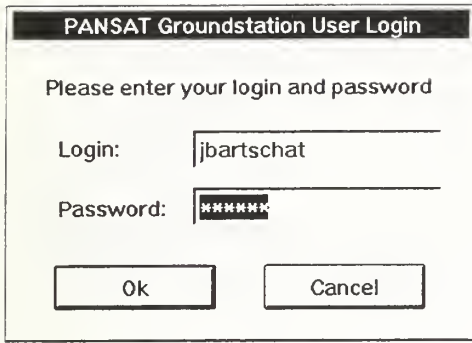
Figure 14: The Login Dialog

The `Login` dialog as shown in Figure 14 must be invoked before any communication with PANSAT may be established. Both a user login and a password must be provided. An internal user list then recognizes the user status. The groundstation application divides into four of them:

System Administrator (all functionality, including password setting for other users), *Super User* (all PANSAT-related functionality), *Intermediate User* (all PANSAT-related functionality, except `Low-Level` and `Memory` dialog features), and *Normal User* (only `Mail` dialog features).

IV. DEVELOPMENT PREPARATION MANUAL

This preparation manual is intended for use by a developer who wants to continue developing either the groundstation software or create an application using a similar environment. It consists of three sections. The first section describes the necessary steps of installing and configuring the various software tools related to the development of the PANSAT Software Groundstation. The second section contains a brief explanation of basic programming knowledge necesssary to understand not only how the groundstation software is structured, but also how the Windows operating system related applications are implemented in general. The third section finally puts the first two together and explains briefly, how the various software tools may assist you in developing an arbitrary application taking the groundstation software as an example.

The following will be very useful for programmers who want to develop their own applications in a similar environment to the one used for the PANSAT Software Groundstation, which could be considered as a typical Windows application. It assumes you are already familiar with at least the "Groundstation Requirements" chapter.

A. INSTALLATIONS AND CONFIGURATIONS

This section describes the necessary steps of installing and configuring the various software tools related to the development of the PANSAT Software Groundstation.

1. Basic Installations

After successfully installing Windows NT 3.5 and MSVC 2.0, WinWidgets (WW) and Resource Workshop (RW) were installed. Following the automated installation, these tools could still not interact. The following steps describe how to access RW from within MSVC, and how to tell MSVC, RW and the current application to work properly with WW.

2. Connecting the Compiler to the Resource Workshop

Start MSVC. Under the `Tools` menu item, choose `Customize...` and the tabbed dialog `Tools`. Enter the path to the Resource Workshop as you installed it on the harddisk under the `Command` entry, or browse for `Resource.exe`. Enter `Workshop` or another more convenient name under the `Menu Text` entry. Under `Arguments`, enter `$RC` and every time you start RW, the current *.RC file containing resource information from your application will be passed to RW and automatically displayed by it. Under `Initial Directory`, enter `$ProjDir` and RW knows that the current resource file could be found in the project directory created by MSVC. For a more detailed description of all possible features, press the Help button.

Now affirm all your actions and leave the `Tools` dialog. The `Tools` menu bar should now contain the new entry `Workshop` (or any other text you wrote into the `Menu Text` section of the `Tools` dialog).

3. Connecting the Compiler and the Application to WinWidgets

Once you have created an application skeleton with MSVC, you are ready to connect MSVC and this application to WinWidgets. If you do not know how to create an application skeleton, refer to "Programming with the Microsoft Visual C++ Compiler".

If you plan to use WW with more than just one application, it makes sense to put all necessary changes in a text file. This file could look like the following:

27

```
C/C++ Preprocessor AND Resources:
d:\widgets\include,d:\widgets\cpp\include

Link Input:
d:\widgets\lib\xtbl32.lib d:\widgets\lib\widge32.lib

InitInstance():
WidgetsInit();
XTableInit();

App.h:
#include "mfcwidg.h"
#include "mfcxtbl.h"
```

Load this file into a text editor or to a temporary opened new text window under MSVC. The four paragraphs contain all the information you need to paste into MSVC settings and/or your application. This file assumes that you have installed WW in the `Widgets` drawer of your `D:` drive. If not, change the appropriate lines in the above text.

Invoke MSVC and the `Project/Settings...` dialog. Choose the `C/C++` tab and from the appearing `Category` drop list, choose `Preprocessor`. Copy the line following "C/C++ Preprocessor AND Resources:" into the clipboard and paste it into the `Additional Include Directories` edit box. Now choose the `Resources` tab in the settings dialog and paste the same string into the `Additional Include Directories` edit box of this tab also.

Now choose the `Link` tab and from the appearing `Category` drop list, choose `Input`. Copy the line following "Link Input:" into the `Object/Library Modules` edit box.

Open the application file of your previously created project. If you named your project `XYZ`, then the application file of your project is named `XYZ.cpp` and could be found in the drawer `XYZ` (unless you specified otherwise; this is the default setting). In `XYZ.cpp`, you find a method called `InitInstance`. Copy the lines following "InitInstance():" at the beginning of the `InitInstance` method.

Open the corresponding include-file, that is, the include file named XYZ.h. It contains application-wide declarations and definitions and is included in every *.cpp file

of your project. Copy the lines following "App.h:" (this is meant to refer to XYZ.h, in this example) at the beginnig of XYZ.h. If you use precompiled headers[3], you can copy these lines into your **stdafx.h** file also (automatically created by the compiler).

4. Connecting the Resource Workshop to WinWidgets

Invoke Resource Workshop by choosing **Tools/Workshop** from within the Visual Workbench (the MSVC IDE[4]). If you want Resource Workshop to recognize WinWidgets, it must be told how to display the controls provided by WinWidgets, and how to work with them. A so-called control library ships wiith the WinWidgets package. It is called **hdlg.dll** and contains all visual information about the controls in a dialog editor readable format. From within Resource Workshop, choose **Install Control Library** from the **Files** menu after creating a new dialog with **Resource/New/Dialog**. The **Tools** floating dialog should now be extended by a couple of new entries. Choose **File/Add To Project** and include the file **mfcwidg.h** from the **widgets/cpp/include** directory (your WinWidgets installation).

B. PROGRAMMING WITH THE MICROSOFT VISUAL C++ COMPILER

This section contains a brief explanation of basic programming knowledge necesssary to understand not only how the PANSAT Groundstation Software is structured, but also how Windows operating system related applications are implemented in general. It assumes you are familiar with C and C++ programming.

---

[3] Precompiled headers contain parts of your application which are seldomly changed, but contain frequently used definitions. When building your application, these precompiled headers are included without recompiling, thus speeding up the edit-debug turn-around process. Use project settings to use this feature.

[4] IDE: Integrated Development Environment

This description refers to the Microsoft Visual C++ Compiler version 2.0 for use under Windows NT. It is still valuable for use with the previous compiler versions which run under Windows 3.x; however, a few adjustments according to menu or dialog references should be made in this case.

1. General

Programming with MSVC most likely means programming using Windows-specific environment (in case of MSVC version 2.0 or more, it means programming using Windows NT). As Windows NT is a graphical oriented operating system, the whole application development procedure falls into two parts: first, the definition of the GUI and second, the actual programming of the controls offered by the GUI.

The first part (definition of the GUI) means painting and designing the outward appearance of your application with a tool called a dialog editor. MSVC has a built-in dialog editor named AppStudio, and there are several other dialog editors on the market, such as Borland's Resource Workshop which we are using to design applications. Their input and output are text files containing descriptions of the controls, menu bars and dialogs you chose to design. These text files are compiled by MSVC (or rather any Windows based development system) and added to the compiled C/C++ files of your application.

This leads to the second part (programming of the controls offered by the GUI) of the application development. The GUI itself is performs no useful action; it does not contain any intelligent user interaction code. This interaction code is what you have to write in order to get your application working. The GUI is just the outward appearance; the data structures and meanings of mouse clicks, menu choices and so on is up to the application's developer. This is exactly what you do when writing C++ code for an application. You build data structures to store internal data just the way you would in a non-Windows-based program. You refer to GUI parts (controls, menus, accelerators,

dialogs,...) via **#define**'d integer values[5]. Windows itself offers many functions to choose, change, and draw GUI parts. You use these functions (or the corresponding MFC function, see below) with the defined integer values as parameters to address the control you want.

2.   Microsoft Foundation Classes (MFC)

The Microsoft Foundation Classes is a class library that encapsulates Windows functions in special classes. Programming with MSVC primarily means programming with MFC. This is just a thin layer above Windows, but it simplifies and and structures the application development. Almost every function (or, spoken in C++, method) used in the groundstation is a MFC function. If you know how to program MFC, you know how to program Windows - and even a little bit more.

Windows (NT) is a message-based operating system. That means, program code is not executed from beginning to ending; instead, the methods the application contains are invoked because Windows invoked them, by sending messages to your application. And Windows sends messages because the user sitting in front of the computer clicks or writes something within the application. From the programmer's view, most of this message passing is hidden; fortunately, because even without worrying about message passing, Windows programming is difficult. This message passing is hidden in MFC (but conducted by Windows, as MFC is only a layer above Windows for programmer's convenience), and to make use of it, MSVC is provided with a tool called ClassWizard (see "Using ClassWizard to Change Your Application" for details).

---

[5] Found in the **Resource.h** file of an application.

3. The Document-Frame-View Architecture

If you already have done serious programming, you might have encountered a problem not easy to deal with: your application grew too large to maintain a proper overview. And if you continue to increase the code, you start spending more and more time just looking for previously programmed code in order to find out about the correct software interface definitions you invented.

As Windows-based applications grow large quickly, MFC offers a built-in order scheme in shape of the so-called *document-frame-view architecture* (sometimes referred to as *document-view-* or *doc-view-architecture*). The three parts represent three different sections of your own application as far as the programming contents is concerned. By creating an application skeleton with AppWizard, this threefold architecture is implemented automatically. MFC uses special classes to represent each part of this architecture. You REALLY should make use of it.

A **document** contains all the data structures, calculation methods and other internal processing routines representing the core of your application. It does *not* know anything about user interaction or GUI programming. It is what your program is all about; the document represents just pure functionality. It is normally derived from the MFC `class CDocument`.

A **frame** is all visible controls, like scrollbars, buttons, edit boxes, checkboxes, listboxes and comboboxes, all menu bars and dialogs and their controls. In short: it is all you can click on and expect some kind of action from. As its name implies, it is a frame for the application, its outward appearance. It is normally derived from the MFC `class CFrameWnd`.

A **view** is the so-called client area of a frame. This is the blank middle part of a frame in which you can make your application paint, draw or write what you want it to. It works as the visual interface between the document and the user; thus it normally

displays data stored and/or calculated in the document to the user. It is normally derived from the MFC `class CView`.

Depending on several settings and your own intentions, the main document, frame and view classes may be derived from other base classes of MFC. These details are not within the scope of this thesis.

It is the programmer's responsibility to program the doc-view architecture in the above specified way. Once you have performed a little programming, you should develop a better feeling for how and when to put your methods in one of the above parts. MFC just provides you with a framework; you fill it out by yourself. All of the above mentioned classes can access the two others in a limited way, so be careful where you put your methods and whether they have to use features only supported by another class.

As mentioned above, the view displays document data. Thus, it has a `GetDocument` method to obtain a pointer to the document attached to that view, so the view can access all public data of the document via this pointer. On the other side, the document has an `UpdateAllViews` method which invokes a special method inside every view attached to this document. This method should be called when the document is ready with a lengthy calculation. Upon receipt of this `UpdateAllViews` message, the view can update its display using the data the document just finished calculating. You, the programmer call `UpdateAllViews` from within your calculation routine inside your document class, because you know best when the data is ready to be displayed. The special method based in the view class and invoked by `UpdateAllViews` is named `OnUpdate`. Knowing this, you just override `OnUpdate` in your view class and write a couple of lines of code into this method to take care of displaying the calculated data. This description is only intended to be a short overview of what a message based operating system is capable of and how clean it could be programmed, if you use its capabilities correctly.

4. Project Files

A C++ project (sometimes referred to as an "application" or "program") contains multiple source files bound together. This project can be changed at any time during the development process. You normally create an application with AppWizard, because this is the most convenient way and you do not have to worry about the application framework.

By default, every base class (such as the document, frame and view class) has its own include (*.h) and implementation (*.cpp) files. For example, the groundstation project name is "gnd". You will find the document class definitions in `gnddoc.h`, and the document class implementation in `gnddoc.cpp`. Same with the view: `gndview.h` and `gndview.cpp`. The (one and only) frame resides by default in `mainfrm.h` and `mainfrm.cpp`. In addition, a fourth pair of files is generated: the application file in which the document, frame and view are linked together in a so-called document template: `gnd.h` and `gnd.cpp`.

So far, only the code part of the application is concerned. What about the visual part, all the controls, dialogs etc.? By default, a file named `gnd.rc` is generated. It contains all information about controls and dialogs, where they are placed and their properties. This file must conveniently be edited in a visual manner (although it is a text file) with a dialog editor, in our case Resource Workshop. In addition, a separate file named `Resource.h` contains all the `#define`'s for all resources[6], that is, their integer Ids.

---

[6] Resources are controls and dialogs. Controls are scrollbars, buttons, checkboxes, radiobuttons, listboxes, comboboxes and other custom controls, such as grids, spin controls, and so on. Dialogs can contain controls, and they can be ordered in so-called tabbed dialogs (used by the groundstation).

C.   TOOL'S REFERENCE

This reference describes the main tools a software developer will be working with using MSVC. Their use makes programming more convenient; thus allowing the developer more time to concentrate on the actual application details instead of the framework's. Parts of this array of convenient tools are the MSVC AppWizard and ClassWizard as well as the add-on Borland Resource Workshop. Finally, the MSVC Online Help as a necessary source of all types of development-related information, is most valuable.

1.   Using AppWizard to Create an Application Skeleton

Windows based MFC applications require a huge framework overhead before doing anything useful. To prevent the programmer from re-inventing the wheel for each application, MSVC is provided with AppWizard (invoked by `File/New/Project`). It is used only once during the development: at the very beginning. AppWizard lets you define the desired features of your application by clicking in its dialogs. Refer to the Installation Manual or Online Help for details. Most of the choices are self-explanatory. However, there are several things you must be familiar with before you start AppWizard and program. You can easily try and see what the output files look like by invoking AppWizard and choose arbitrary settings. It takes just a couple of seconds to create an application skeleton, and everything is put into a separate subdirectory, so you can easily delete the entire project with the Windows File Manager.

Single Document Interface (SDI) means that there is only one document class and normally only one view attached to that document in your application. Multiple Document Interface (MDI) creates an extendable document template, thus more than one document class can be used, and usually more than one view can be created, too. With groundstation, the SDI concept was used.

You can also choose OLE[7] capabilities as well as database support via ODBC[8] compatibility. These are advanced topics you should not use unless you fully understand what they support.

2.   Using ClassWizard to Change Your Application

ClassWizard can be used to maintain and change classes, use the message map system to link visual controls to program code or change class member variables. It can also be used for OLE related maintenance, which is not described here. The most essential and often used task with ClassWizard is its message map maintenance capability. To understand a little bit better how ClassWizard works, you must first learn how to implement the message map system without use of ClassWizard.

As mentioned above Windows (NT) is a message based operating system. Thus, every application programmed for use with Windows and taking advantage of its graphical capabilites, must be able to do message passing from and to Windows. In MFC, this goal is achieved by the concept of **message maps** and **window functions**.

A message map is a set of commands which tell Windows

- which type of message Windows should catch,
- to which visual control the message should be attached to,
- which window function should be executed upon receipt of this message.

---

[7] OLE: Object Linking and Embedding. This technique allows users to link or physically embed (include) objects created from OLE servers into an OLE client application. An OLE client application serves as a container for data created by OLE servers. OLE itself is the standardized interface to allow OLE item interchange by copy/paste or drag and drop.

[8] ODBC: Open Database Connectivity. This standardized interface definition simplifies database record interchange between DBMS (Database Management Systems) as well as ODBC compatible C++ applications (for example). It requires ODBC drivers for both sides of the interchange chain, which are available from third party developers.

It looks something like the following:

```
BEGIN_MESSAGE_MAP(CGndView, CView)
  //{{AFX_MSG_MAP(CGndView)
  ON_COMMAND(ID_ACCESS_LOGON, OnUserAccess)
  ON_COMMAND(ID_ACCESS_LOGOFF, OnEndUserAccess)
  ON_COMMAND(ID_PREFERENCES, OnPreferences)
  //}}AFX_MSG_MAP
  // Standard printing commands
  ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
  ON_COMMAND(ID_FILE_PRINT_PREVIEW, View::OnFilePrintPreview)
END_MESSAGE_MAP()
```

The **ON_COMMAND** macro tells Windows that the type of message is a command message. The first parameter refers to the visual control whose activation you want to catch. For example, the above **ID_ACCESS_LOGON** identifier (as defined in **Resource.h**) refers to a menu entry in the main menu of the groundstation application. And every time the user activates the menu entry with this specific ID, the method **OnUserAccess** is invoked. This special method is also called **window function** or **message handler**. It is part of the class for which this message map is defined, in this case **CGndView** which is derived from the MFC class **CView** - the former being the view class of the groundstation. This information is given by the parameters of the **BEGIN_MESSAGE_MAP** macro. The whole message map as shown could be found in the implementation file for the application view, **gndview.cpp**. Every class containing a **DECLARE_MESSAGE_MAP()** macro in its class declaration (to be found in **gndview.h** for this example) can contain a message map of the type shown above.

Besides the entries into the message map, there are two additional things to remember in order to make the message passing mechanism work: the declaration of the window function in the class declaration file, and the definition of the window function in the implementation file.

The declaration of window functions could look like this:

```
protected:
// Generated message map functions
    //{{AFX_MSG(CGndView)
    afx_msg void OnUserAccess();
    afx_msg void OnEndUserAccess();
    afx_msg void OnPreferences();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

All window function declarations start with `afx_msg`. Right now, this `#define` resolves to void; however, future versions of MFC may redefine its meaning. No window function does return a value, and only some take parameters. This excerpt is located in `gndview.h` in the `CGndView` class declaration.

The definition of the window function looks somewhat like this:

```
void CGndView::OnUserAccess()
{
  // Put your code handling the activation of the Access/User
  // menu item from the groundstation main menu here
}
```

Conclusion: add a control item handler to your application by

- adding an appropriate entry into the message map of the appropriate class,

- add a window function declaration to the class declaration,

- write the window function into the implementation file of the appropriate class.

In most cases, you do not have to remember those steps nor the changes you have to make to the *.h/*.cpp pair of files of the appropriate class. Instead, you use ClassWizard (`Project/ClassWizard` and the `Message Maps` tab from within the Visual Workbench). You see all classes recognized by ClassWizard available in the `Class name` combobox. Choose the appropriate class; if you do not know what the appropriate class would be, review "The Document-Frame-View Architecture" discussed

above, or refer to appropriate help files within the MSVC Online Help. Then choose the ID of the visual control you want to create a window function for from the `Object IDs` listbox, and the message type in the `Messages` listbox. Normally you only have the choice between `COMMAND` and `UPDATE_UI`, and most of the time `COMMAND` is chosen. You specify the name of the window function and click `Add Function`; and all of the above changes are made automatically.

However, sometimes ClassWizard does not recognize all classes. Either rebuild your ClassWizard database by deleting the *.clw file (i.e., `gnd.clw`) first, then invoke ClassWizard and follow the instructions, or implement the necessary lines of code yourself.

3.   Using the Resource Workshop Dialog Editor

Borland's Resource Workshop (RW) is an advanced dialog editor[9] for Windows or Windows NT and is intended to be used with any software development application which understands and implements the commands defined for Resources (extension *.rc). An *.rc file (**resource file**) is a text file that can be edited with any text editor; any changes, which are in compliance with the Resource Language, made to this kind of file thus will be recognized correctly the next time RW processes it. However, this is not the recommended way to change your resource file. RW allows changing these files in a graphical manner. If you want to learn about editing a resource file, read the instructions in the help files provided with RW. Nonetheless, a short list of information is provided which you should be aware of before you actually should use RW, and for troubleshooting using RW in connection with AppStudio.

---

[9] Dialog Editor: every software tool which is able to edit a resource file in a visual manner. This includes not only *dialogs*, as the name might suggest, but *every* kind of resource type and visual control.

The normal working procedure of a dialog editor is a useful thing to know, because in case of errors you will be able to tell more quickly where this error might have occurred. This description is not limited to RW; other dialog editors work similarly.

When starting up RW, either the corresponding resource file is loaded automatically (from within the Visual Workbench, if you have defined the appropriate parameters), or you have to open or create a new one (when using RW as a stand-alone dialog editor). But first, what is a **resource**? Every kind of data which normally refers to the *outward appearance* of an application is stored as a resource. It can be edited and compiled separately from the application code, and it can be loaded to or discarded from memory at any time the user invokes certain resource-based visual controls or the operating system needs memory. When working with large applications on a small computer system, you might have encountered a delay and harddisk access when you clicked on a menu or invoked a dialog. In this case, the operating system discarded former instances of these resource types from memory and had to load them again at the time you needed them.

You are able to create and edit all kinds of **resource types**, such as *dialogs*, *accelerators* (key-invoked commands), *menus*, *bitmaps*, *cursors*, *fonts*, *icons*, *stringtables* (lists of strings you wish to put in resources) and a couple of others not commonly used. *Dialogs* are the most complex of these resource types because they can be related to all of the above types and the **visual controls**. Visual controls in general are all kinds of visual gadgets you expect a reaction from when clicking them with the mouse pointer. Thus, **visual controls** are *pushbuttons*, *checkboxes*, *radiobuttons* and a couple more button derivatives, *listboxes*, *comboboxes* (drop-down listboxes), *statics* (statically displayed text), *editfields* (editable text) and, in case of the WW-package, many more sophisticated visual controls, such as *grid controls*, *spin controls*, *tab controls*, *toolbars* and *spreadsheet controls*. To learn more about those visual controls, refer to the WW help files and [Ref. 1].

As every resource type and visual control can be customized (which will be done in most of the cases), the programmer has to define their **properties**. This is done inside RW by just double-clicking the desired resource and define or alter the parameters presented in the upcoming properties dialog box. In case of the WW *button*, for example, the programmer first has to decide whether to use a *pushbutton*, a *checkbox* or *radiobutton* (which are *buttons* the way WW understands it), a *tristate* or *grouped button* (two or more buttons linked together, of which only one can be pressed down). Then he can go on defining the appropriate text, icon, flags and other parameters needed to display the visual control correctly.

Every visual control has its own properties. Refer to the appropriate help files and reference manuals for further information, or just use a trial-and-error strategy as most of the property definitions (flags, positions, entries, height, width, sub-controls,...) are self explanatory.

Most dialog editors (and RW is not an exception) use two files to store their information. First, in an *.rc resource file the location and properties of resource types and visual controls is encoded in Resource Language. Second, a *.h resource header file (in case of MSVC, `resource.h`) contains all necessary `#define`'s which serve as an interface between the resource file data (from the *.rc file) and the code added by the programmer to access this data from within the C++ code (from within the C++ source files). The programmer defines **control names** for every visual control he creates using RW. By convention, they are all uppercase, starting with `IDC` (*control identification*) with underscore keys to separate the descriptive parts. These definitions are converted to `#define`'s in `resource.h` as integer values. The programmer refers to a visual command inside his C++ source by using the control name defined in RW, and the compiler resolves to the appropriate integer value because of the entries in `resource.h`.

Conclusion: What you basically do with a dialog editor is
- load or create a **resource file**,

- choose the desired **resource type** and define its **properties**, and, in case of a dialog,

- place **visual controls** wherever you want them to appear, defining their **properties** and often customizing their width and height as well as the alignment to other visual controls, and finally

- saving the edited resources to your resource file.

All steps following this process (compiling the resource file, including it into your application) are done automatically by the MSVC compiler and linker and thus are independent from RW. The "only" remaining is actually programming the RW-defined resources. What you created with RW is just the outward appearance of your application, but no functionality except the default visual control behaviour[10] is implemented. The programmer himself has to take care of how every visual control behaves and what reaction it produces. He can rely on the classes provided with MSVC and WW in order to access, control and alter every visual control. This makes clear that the actual programming has to be done after the resources have been designed with a dialog editor.

RW normally ships with the Borland C++ Compiler, but can also be used as a stand-alone application. This makes it valuable for use with MSVC, the biggest competitor for Borland's C++ compiler on the market right now. The current RW version 4.5 allows basically all operations which would be possible to perform with the MSVC built-in AppStudio. However, in contrary to AppStudio the RW Dialog Editor allows installation of additional control libraries, which is a necessary feature to use the

---

[10] Default visual control behaviour: every action done automatically with a visual control by the framework. Examples: clicking on a checkbox or radiobutton alters the state and visual appearance from "not crossed" to "crossed" ("not bulleted" to "bulleted" in case of a radiobutton) or the other way around, respectively. Using a combobox, the framework takes care of highlighting clicked entries, dropping down and displaying entries as well as scrolling them with an associated scrollbar; the programmer has to provide the text of the entries.

WinWidgets Custom Control package. This is a big advantage of RW and the reason RW was chosen instead of AppStudio.

But there are also several disadvantages using RW a programmer must be aware of, especially when he is already used to working with AppStudio. As correct handling requires only minor effort, RW is still the appropriate choice.

RW does not support automatic control name numbering as supported by AppStudio. Thus, the programmer has to check frequently the integer values as `#define`'d in `resource.h`. You should not use integer values more than once, because the compiler will resolve to the visual controls according to these values. You might lose track of which value you already used. Furthermore, you will have to use AppStudio as well as RW[11], and thus have to adjust manual numbering as necessary with RW to automatic numbering as supported by AppStudio. This basically limits you to *which* integer values you can use; refer to the AppStudio User's Guide provided with the MSVC help files for further information, and check some `resource.h` files as reference.

It also might happen that RW complains about AppStudio-processed *.rc resource files. There are two possible reasons for this. The first is due to an AppStudio or RW bug: several lines in the resource file are copied twice into the edited file, thus making correct processing impossible. Just delete those lines with a normal text editor; RW will complain about this error with a line number as reference. Second, a `#define` used and sometimes discarded by AppStudio may prevent RW from correctly processing it. Just `#define` it again in the resource file, and RW will process just fine.

---

[11] This will happen when you have to rebuild the ClassWizard database (*.clw) in order to let it recognize non-MSVC classes (as used with WW). In this case, you will have to invoke AppStudio prior to ClassWizard, because ClassWizard's database relies on correct resource file entries. Sometimes processing errors might occur when invoking a RW-edited *.rc resource file from within AppStudio. Most of the times this is because of some additional *.h header files were not included in an AppStudio-conform way.

4. Using the Online Help and the Contents Browser

As development platforms such as MSVC are very complex applications, hardly any programmer can memorize all classes, their methods, their parameters and return values (there are a couple of hundreds of them). Either a programmer has kilograms of books right beside him while programming, or he makes use of the built-in online help capability. In fact, the most convenient way is a combination of both.

There are two kinds of information you might want to obtain while developing an application. First, information about Windows functions, C/C++ features and other things already shipped with the compiler. Use the **Online Help** for that information. Second, your self-programmed variables, structures, classes and methods. For this kind of information, use the **Contents Browser**.

Invoke the Online Help with the F1 key while the cursor is placed over a method or a defined variable, a C/C++ keyword or any other code you expect to be considered in the help database. However, Visual Workbench will tell you very soon whether it recognizes the data beneath the cursor or not. For example, if you want to know more about the class `CString` (which could be completely unknown to you now, but which you really should try to learn more about), just place the cursor somewhere in the word `CString` and press F1. Another way to access the online help is via the `Help` menu. You can read all information available on paper for MSVC with `Books Online` (if you have the CD version of MSVC 2.x). Other useful information can be found in the various Hierarchy Charts, which are a good place to start for general information about MFC and its classes. The online help will become a very helpful tool for you while developing your application.

Naturally, you will not get a full description of your *own* code unless you wrote a couple of lines about it. What you can get from the Visual Workbench is the location of

the declaration[12] and the definition[13] of your code. Use the F11 key from within the Visual Workbench while the cursor is placed over the code you want to obtain the declaration from, or Shift-F11 for the definition. Every time you compile your application, a new browser database is created to reflect the latest changes to variable declarations and definitions. You can switch this feature on or off in you Project settings (**Project/Settings...**, tab **C/C++**, checkbox **Generate Browser Info**).

---

[12] Declaration: letting the compiler know of what type the variable is. Does not use any memory. Example: **int i;** declares a variable named **i** as of type **int**. The Visual Workbench uses the expression "Definition" for that matter.

[13] Definition: assigning a value to a variable which previously has been declared. Uses the amount of memory the variable type uses. Example: **i = 100;** allocates 2 bytes (under Windows 3.x) or 4 bytes (under Windows NT) of memory to store an integer variable of value 100 in that memory, and letting the programmer refer to it as **i**. The Visual Workbench uses the expression "Reference" for that matter. C and C++ allow mixed declarations and definitions: **int i = 100;**.

V.   PROGRAMMER'S REFERENCE

This reference describes the programming of the PANSAT Software Groundstation. It contains a much more detailed description of the actual procedure to program Windows or Windows NT with C++ than other chapters. Thus, you should already be familiar with all preceeding chapters, programming in a high-level language, especially in C or C++, and exhibit an advanced knowledge about graphical oriented operating systems, commonly used data structures and programming techniques.

A.   THE GROUNDSTATION DOCUMENT

The groundstation document is not only placed in the `CDocument` derived class of the `gnd` application, but expands to some other structures whose implementation could be found in the `gnddoc.cpp` file also, as they logically belong to the application's document. Because all of these additional structures must be accessible from every class of the application, they are defined globally and statically[14] in the `gnd.h` file (which is included into every *.cpp file created by AppWizard by default).

1.   Classes and Structures

A **class** encapsulates data and functions (they are called methods when they are a part of a class) in one structure. This is a simple, but very effective way to keep data and functions accessible only from where an access makes sense, that is, only within the class for which they are defined. A **structure** in the original meaning is just an encapsulation of data, that is, variables and/or substructures. The C++ language expands

---

[14] Static variables only use **one** memory location, no matter how often they are declared and defined. This ensures that every method using this variable uses exactly **this** variable and not a locally defined other variable with the same name. Static variables use the C/C++ `static` keyword before their type identifier: `static int i;` declares a static variable named **i** of type integer.

this meaning allowing C++ `struct`'s to be equivalent to a class: not only data members, but also functions (methods) can be encapsulated in structures.

The only difference between `class`'es and `struct`'s is the default access behaviour. Certain C++ keywords determine the accessability of data members or methods (furthermore referred to as "members"): `public`, `protected`, and `private`. They determine whether non-class methods can access class members, and how they are accessed in derived classes. Refer to the online help for further information.

The most important structures belonging to the document are not part of the `CDocument` derived class `CGndDoc` because they deserve unique structures. These are the PCL output structures `struct SMacro`, `struct SCmd` and the PCL input structure `struct SReturnCmd`. They contain all necessary members to cope with the problem of sending, receiving, storing and loading data from and to disk, and from and to PANSAT via the serial interface. Furthermore, all necessary information about PCL is stored in the **program command database** inside the definition of `struct SDocumentCmd`. Both the structure declaration `struct SDocumentCmd` and its definition (named `DocCmd`) can be found in `Gnd.h`, like all the declarations of the structures mentioned above.

2.   The PCL Output Structures

The PCL output structures `struct SMacro` and `struct SCmd` are based on the information given by the program command database stored in `const static SDocumentCmd DocCmd[]`. In this chapter the following questions will be answered:

- What is a macro, what is a script? What is a command?

- What are the contents of the program command database?

- How does PCL output work with the macro and command structures?

47

a. The macro/command relationship

A **command** is one (1) PCL command[15] and its parameters, if applicable. A **macro** is a sequence of commands. A `script` is a sequence of commands including at least one `script command`[16]. The C++ structures used for representing this relationship are `struct SMacro` and `struct SCmd`:

---

[15] PCL command: every command recognized in the program command database which represents a command to PANSAT.

[16] Script command: every command in the program command database which is not intended to be sent to PANSAT, but controls the groundstation software. No script command is implemented yet.

```cpp
struct SMacro
{
public:
  SMacro();
  SMacro(int nIndex);
  SMacro(const char *pName);
  virtual ~SMacro();
public:
  virtual int Load();
  virtual int Save();
  virtual int Overwrite();
  virtual int Execute();
  virtual int GetError();
  BOOL IsScript();
protected:
  BOOL m_bHasChanged;
  int nError;
private:
  const char *GetFileName();
  BOOL SetFileName(const char *pName);
  const char *GetMacroName();
  void SetMacroName(const char *pName);
private:
  CString FileName;
  CString MacroName;
  CPtrArray cmd;
  BOOL m_bHasFileName;
  BOOL m_bIsBuiltIn;
  BOOL m_bIsScript;
friend class CCHScriptsDlg;
};

struct SCmd : public SMacro
{
public:
  SCmd();
  SCmd(int nIndex);
  ~SCmd();
public:
  __int8  cmd;
  __int16 wParam;
  long    lParam;
  void    *ptr;
public:
  virtual BOOL Load(void *fh);
  virtual BOOL Save(void *fh);
  virtual BOOL Execute(void *fh);
protected:
```

```
    virtual long GeneratePassword();
    BOOL ReadString(void *fh, long *pLong, int *pnErr);
    BOOL WriteString(void *fh, char *pchar, int *pnErr);
};
```

SCmd is derived from SMacro and thus "knows" about everything what is
derivably defined in SMacro. The SCmd structure has to comply with several
requirements:

- Store every possible command in an identifiable manner,

- Provide storage capability for every possible parameter or combination thereof for
  every possible command,

- Provide disk I/O functionality to save or load one (1) command from or to disk,

- Feature serial output for one (1) command.

These goals are achieved by the various members of the SCmd structure.
Furthermore, it contains several more members to construct a command, generate a
password (if applicable) and simplify disk access. The four members cmd, wParam,
lParam and ptr serve as storage for every possible command (member cmd) and its
parameters (members wParam, lParam and ptr), the latter of which represent data types
and structure pointers according to the entries in the program command database (see
"The Contents of the Program Command Database").

Disk I/O is done by the methods Load and Save; command execution (which is
very similar to disk I/O functionality except that data is sent to COM1) is performed by
Execute. These three I/O methods require an already opened I/O channel whose handle
they take as the only parameter (void *fh, fh: File Handle). This task is completed by
the SMacro structure described below. The file routines used here are part of Windows
and Windows NT. For more information about programming with these routines, refer to
"The Implementation of the Output Structures" discussed later in this chapter.

`SMacro` is the more general one of the output structures. It takes care of several file and data channel maintenance tasks:

- Opening and closing the I/O channels for disk I/O and/or serial output,

- Provide storage for the filename associated with this macro and the macro name and means to change those names,

- Prevent the user from involuntarily erasing altered macros,

- Provide the programmer with an easy error handling capability,

- Provide means to store a virtually unlimited amount of commands (each represented by a `SCmd` structure).

These goals once again are achieved by the various members of `SMacro`. The two disk related methods `Load` and `Save` use the Windows Common Dialog Box feature to present dialog boxes to open an I/O channel for loading or saving data from or to disk. Refer to [Ref. 2] for further information. The actual load or save procedure then can be conducted by invoking the `Load` or `Save` method of `SCmd`; this is done for every command that this macro contains, so the whole sequence of commands can be loaded or saved. The `SMacro`-method `Execute` works similar to that: it just opens the I/O channel to COM1 and leaves the rest to the `SCmd Execute` method.

`SMacro` owns two data members in which the macro name and the filename of the macro are stored: `CString MacroName` and `CString FileName`. This class provides the programmer with a dynamically length-adapted string storage as well as many useful methods for string handling and conversion. The `CString` class as part of the Microsoft Foundation Classes (MFC) General Purpose Classes is subject to a closer discussion later in the text. The four `SMacro` members `GetFileName`, `SetFileName`, `GetMacroName` and `SetMacroName` offer easy methods for the programmer to retrieve a string (`Get...` members) from the user and store it into the `CString` data members `MacroName` and `FileName` (`Set...` members).

The BOOL member m_bHasChanged should always be set to TRUE everytime the macro has been changed. The disk I/O methods Load and Save as well as the class destructor check for this boolean variable prior to erasing operations and thus prevent from involuntary data loss. The programmer is responsible for setting this variable to the appropriate value every time edit or storage actions have occurred.

Many errors might occur during I/O actions. Especially the Load, Save and Execute methods contain a multitude of I/O actions; thus, they contain several error checks. The integer return values of those functions refer to the ErrAry text array (Gnd.h). They represent the zero-based index of the string entries in this text array.

So far, no members of SMacro refer to a PCL or script command or their parameters. This is done on purpose because the SCmd structure already covers that whole problem. As multiple commands can be part of a macro, SMacro contains the CPtrArray cmd member to provide means for storage. It provides a dynamically growing (or shrinking) array of pointers to whatever you want pointers to (to SCmd structures in this case). The CPtrArray class is part of the MFC Collection Classes and subject to later discussion.

b.   The Contents of the Program Command Database

The program command database is located in Gnd.h and consists of the structure declaration

```
struct SDocumentCmd
{
  char   *command;
  __int8 cmdID;
  int    flags;
  int    wParam_Type;
  int    lParam_Type;
  int    return_Type;
};
```
and the structure definition (excerpt)

```
const static SDocumentCmd DocCmd[] =
{
  {"add_command",   0x01, FPCL|FSUPER|FPASS,
          TVOID | TQ_CMDPTR,     TL_UTC,     TVOID},
  {"add_task",      0x02, FPCL|FSUPER|FPASS,
          TQ_TASKPTR,            TVOID,      TVOID},
  {"boot_rom",      0x03, FPCL|FSUPER|FPASS,
          TVOID,                 TVOID,      TVOID},
  . . .
};
```

The six parameters of this structure contain all information necessary for the I/O classes to perform successfully. The `char *command` member contains a pointer to the plain command string, the `__int8 cmdID` member (an eight-bit integer, Microsoft-specific data type) the according command identification value. The third member `int flags` is a value of OR'able `#define`'s also found in `Gnd.h`: all those commencing with an `F`. The last three members contain type information about the variable which is about to be stored in the `SCmd` members `wParam` and `lParam` as well as the `SReturnCmd` member `ptr`. The appropriate `#define`'s all start with `T` indicating a type identifier. Refer to the comments added to the source text in `Gnd.h` for further information.

c.   The Implementation of the Output Structures

The most important thing about the output structures `SCmd` and `SMacro` are the I/O function calls. The structure implementation resides in the `GndDoc.cpp` source file. You can find out more about the used methods in [Ref. 3] and [Ref. 4]. However, first you want to learn more about `CreateFile`, `ReadFile` and `WriteFile` first.

At first glance, the `SCmd` member functions `Load`, `Save` and `Execute` look identical. However, there are a few differences. The disk I/O methods do not have to save or load passwords because they are only needed when uplinking commands to PANSAT. Therefore, only the `Execute` method uses password generation via a `GeneratePassword` method call. As one would assume, `Load` uses `ReadFile` calls, whereas `Save` uses `WriteFile` calls. For easier loading and saving of strings from and

to disk, the SCmd structure features `ReadString` and `WriteString` methods which are invoked similar to `ReadFile` and `WriteFile`.

All `case` branches refer to the according entries in the program command database. Any addition or change to it might also require changes to `SCmd` structure members `Load`, `Save` and `Execute`. A database-independent implementation would have been too complicated.

For some commands it is necessary to provide an additional structure as parameter whose pointer is stored in the `SCmd` member `ptr`. Therefore, all type definitions in `Gnd.h` starting with `TQ_` define which structures may be used for this purpose. So far, the `SCmd` structure itself can be used (for the `add_command` PCL command) as well as the yet to be defined `STask` and `SOSParams` structures.
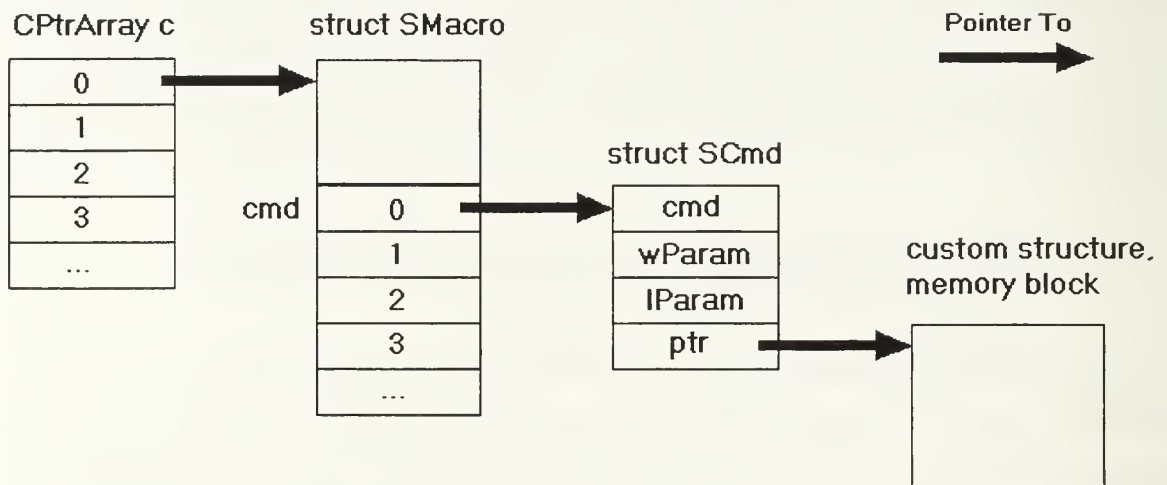
Figure 15: Implementation principle of the SMacro and SCmd structure

Figure 1 shows how the SMacro and SCmd structure is implemented into the rest of the application environment. The highest level of the hierarchy consists of a `CPtrArray` in which pointers to `SMacro` structures are stored. The `CDocument`-derived class in groundstation contains two of them:

- `CPtrArray c` (for command). This `CPtrArray` contains the pointers to all `SMacro` structures necessary to hold all PCL commands and their parameters which are

accessible from within the groundstation application. Currently, every groundstation accessible macro consists of just *one* command, but as it is a **SMacro** macro, it could hold *multiple* commands with no additional programming necessary.

- **CPtrArray m** (for macro). This class contains the pointers to all **SMacro** structures used for the *macro feature* of the groundstation. This feature enables the user to define and execute-by-click commonly used macros.

As described above, the **SCmd** member **ptr** points to a memory block or structure as defined in the program command database by the type qualifier **TQ_** - **#define**'s (located in **Gnd.h**). Because this type qualifier is stored in the program command database, all the programmer has to know is the command ID of the desired command to find out about the correct memory block or structure to be referenced by **ptr**. This determines in all cases how to handle the PCL command, its parameters and return values, if applicable.

3.   The PCL Input Structure

The PCL input structure **SReturnCmd** is declared as

```
struct SReturnCmd
{
    __int8  cmd;
    __int16 size;
    void    *ptr;
};
```

This will allow every data structure to be downlinked from PANSAT not to exceed 65535 bytes. The **__int8 cmd** contains the command identifier according to the program command database and indicates the PCL command whose receipt made PANSAT downlink a portion of data. Before sending the actual data, PANSAT sends the length of that data as a 16-bit-value which can be stored in **__int16 size**. This information, together with the command identification and the program command database, enables the groundstation software (the **SReturnCmd** members, in this case) to determine how the following data portion downlinked from PANSAT shall be

interpreted. The structures **SRFile**, **SRCommandBuffer**, **SRTask**, **SREvent**, **SRTelemetry** and **SROSParams** are declared to serve as containers for that downlinked data. When extending or changing PCL, it might be necessasry to change parts of those structures or add new ones. You should comply to the convention of naming those structures starting with **SR** (Structure Return).

So far, any implementation of the input structure and the return container structures is undecided; the existing structure declarations, however, should serve as a sensible starting point for further development.

Data input must take place in an **interrupt procedure** because the groundstation software cannot constantly poll the COM-port when expecting an answer from PANSAT. In Windows terms, interrupt I/O is called **overlapped I/O**. The well-known **CreateFile** method features overlapped I/O capability. You can find out more about the appropriate methods in [Ref. 4] and [Ref 5]. After reading the overview of these chapters, you should learn more about the **WaitCommEvent** function next.

4.   The Evaluation Process

Because every implemented method cannot completely be evaluated for correct functionality yet nor is all coding complete, the following list might be helpful for programmers who need to continue the programming:

- Check the **SCmd** and **SMacro** structures for correct disk I/O and COM-port access. Refer to the contents of this chapter up to this point for further information and references.

- Implement overlapped port I/O into **SReturnCmd** and check it for correct functionality. You might have to use small additional evaluation programs for this.

- Learn about programming WinWidgets and implement the functionality for all tabbed dialogs. Refer to the "Groundstation User's Manual" for how these tabbed dialogs are supposed to work. Refer to "Using WinWidgets' Tabbed Dialog" and the already implemented functionality for the *Scripts* tabbed dialog for how to program tabbed dialogs and WinWidgets visual controls.

- Learn more about ODBC-compatible programming in order to store all downlinked telemetry in this format. Refer to [Ref. 7] for complete information and [Ref. 8, Chapter 24-27] for a sample ODBC-compatible application.

- Implement additional script language features. Programming the GUI might have given you the appropriate information to continue development.

B.   PROGRAMMING TECHNIQUES USED FOR THE GROUNDSTATION

The following is a short description of important programming techniques which were used to program the groundstation. This description could be very useful both for understanding how the groundstation software has been implemented and as a reference for your own programming. I chose the topics in order to provide an independent and reusable overview of a section of code.
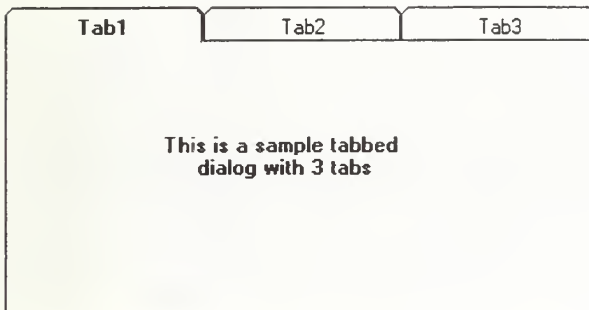
Figure 16: A sample tabbed dialog

1.   Using WinWidgets' Tabbed Dialog

This section describes how to implement the very useful tabbed dialog as part of the WinWidgets Custom Control package. Refer to [Ref. 9] for further information. A tabbed dialog looks like Figure 16. It consists of one outer dialog and three inner dialogs (for this example; one outer dialog may contain more inner dialogs). The outer dialog is derived from the WW class **CTabDlg**, whereas the inner dialogs are derived from **CTabDlgChild**. As a summarization of [Ref. 9], follow these steps to create and implement a tabbed dialog with WW:

- Create each inner dialog and the outer dialog with AppStudio (not RW!) as separate dialogs.

- Use ClassWizard to create a new class for every inner and the outer dialog. Use CDialog as the base class. Refer to [Ref. 10] regarding how to use ClassWizard.

- For the groundstation application view class *declaration* (recommended, `GndView.h` in this case), insert the following:

```
class CGndView : public CView
{
...
public:
    CMainTabDlg        *m_pTabDlg;  // derived from CTabDlg
    CCHScriptsDlg       m_ChDlg0;   // derived from CTabDlgChild
    CCHTelemetryDlg     m_ChDlg1;
    CCHMailDlg          m_ChDlg2;
    CCHMemoryDlg        m_ChDlg3;
    CCHControlDlg       m_ChDlg4;
    CCHOSControlDlg     m_ChDlg5;
    CCHFileSystemDlg    m_ChDlg6;
    CCHTaskControlDlg   m_ChDlg7;
...
}
```

- In each *.cpp source and *.h header file created with ClassWizard, replace every occurrence of `CDialog` with either `CTabDlg` (for the outer dialog) or `CTabDlgChild` (for the inner dialogs). Rebuild the *.clw ClassWizard database file so ClassWizard can recognize the new WW classes.

- Program all dialog characteristics using the `PreSetTabCtrlXXX` and `PreCreateTabCtrlInit` methods according to [Ref.10], if applicable, and encapsulate all dialog specific processing in the appropriate dialog class.

- Add the display code into your view class *implementation* file (recommended; in this case, in `GndView.cpp`). Use the `CTabDlg::AddChildDialog` method to attach every inner dialog to the outer dialog. Invoke the `CTabDlg::DoModeless` method for a modeless display of the tabbed dialog (recommended). Define a view member variable (`m_bTabDlgUp`) to determine whether the tabbed dialog has already been instantiated.

```
void CGndView::OnUserAccess()
{
  if (!m_bTabDlgUp)
  {
    m_pTabDlg = (CMainTabDlg*) new CMainTabDlg(this);
    // add child tabs to tab dialog internal list
    m_pTabDlg->AddChildDialog(CCHScriptsDlg::IDD,
                    (CTabDlgChild *)&m_ChDlg0);
    m_pTabDlg->AddChildDialog(CCHTelemetryDlg::IDD,
                    (CTabDlgChild *)&m_ChDlg1);
    m_pTabDlg->AddChildDialog(CCHMailDlg::IDD,
                    (CTabDlgChild *)&m_ChDlg2);
    m_pTabDlg->AddChildDialog(CCHMemoryDlg::IDD,
                    (CTabDlgChild *)&m_ChDlg3);
    m_pTabDlg->AddChildDialog(CCHControlDlg::IDD,
                    (CTabDlgChild *)&m_ChDlg4);
    m_pTabDlg->AddChildDialog(CCHOSControlDlg::IDD,
                    (CTabDlgChild *)&m_ChDlg5);
    m_pTabDlg->AddChildDialog(CCHFileSystemDlg::IDD,
                    (CTabDlgChild *)&m_ChDlg6);
    m_pTabDlg->AddChildDialog(CCHTaskControlDlg::IDD,
                    (CTabDlgChild *)&m_ChDlg7);
    // fire off tab dialog
    m_pTabDlg->DoModeless(CMainTabDlg::IDD, this);
    m_bTabDlgUp = TRUE;
  }
}
```

The destruction could look somewhat like this:

```
CGndView::~CGndView()
{
...
  // destroy tabbed dialog
  if (m_bTabDlgUp)
  {
   m_pTabDlg->DestroyWindow();
   delete m_pTabDlg;
   m_bTabDlgUp = FALSE;
  }
...
}
```

- There are also three notification messages for handling the creation, activation and deactivation of the child (inner) dialogs. Refer to [Ref. 9] and the TABDEMO sample application provided with the WinWidgets package.

2.   Using WinWidgets' HotLink

Refer to [Ref. 11, *Hot-Linking the WinWidgets to Data* and *Connecting the WinWidgets to Data*] for detailed information. HotLinking is WinWidgets' capability to automatically update certain data types as a result of a user modifying certain visual controls. The advantage of using HotLinking instead of installing a message handler is that it takes less programming effort to settle it. If you wanted to change the contents of a variable according to a user click, you normally would have to:

- write a message handler into a window function,

- attach the window function to a visual control via a message-map entry, and

- expand the class declaration with an **afx_msg** entry of that window function.

For small changes like changing just one's variable contents this would be too much programming effort. WinWidgets allows HotLinking (under certain restrictions depending on the type of the variable) using the WW **SetDataLink** method as shown:

```
WORD m_hlEditMode;

BOOL CCHScriptsDlg::OnInitDialog()
{
...
    CHBRadio  *pEditMode  =
          (CHBRadio  *)GetDlgItem(IDC_SCRIPT_NORMALEDIT);
    pEditMode->SetDataLink(TRUE, &m_hlEditMode);
...
}
```

3.   Using *.ini Files

The concept of *.ini files allows applications to save user-definable data as ASCII-text to a file with an *.ini extension located in the **SYS:\Windows** (or **SYS:\WinNT35**) drawer. This data can refer to a recently used file list, shown dialogs, or some kind of preferences the user does not want to define over and over again every time he launches the application. Thus, use of *.ini files makes application handling easier and more convenient for the user.

Windows offers basically two functions which support *.ini file handling: **GetProfileString** and **WriteProfileString**, both encapsulated in the MFC **CWinApp** class. Refer to [Ref. 12] for a detailed description of these functions. For the groundstation application, this *.ini file feature is implemented as shown in the following code segments:

```
in Gnd.cpp:
CGndApp::CGndApp()
{
  m_pszAppName="PANSAT Ground Station";
  m_pszProfileName="Gnd.INI";
}

in Gndview.cpp:
CGndView::CGndView()
{
  int i;
  strSectionDir    = "Directories";
  strSectionExt    = "Extensions";
  strSectionDscrpt = "Descriptions";

  CGndApp *pApp = (CGndApp *)AfxGetApp();

  for (i=0; i<MAXDIRS; i++)
  {
    PFI[i].Dir = pApp->
              GetProfileString(strSectionDir, def[i]);
    PFI[i].Ext = pApp->
              GetProfileString(strSectionExt, def[i]);
    PFI[i].Des = pApp->
              GetProfileString(strSectionDscrpt, def[i]);
  }
...
}
```

First, Windows has to know about the full name of the *.ini file. The constructor of the application class is a good place to define it (the name is **Gnd.ini** for this application). To retrieve data from that file (assuming we already have meaningful entries in it), the **GetProfileString** method is used. The **strXXX** variables are MFC **CString** class instances in which you can store strings (see below "MFC Class CString"). In order to access the **CWinApp**-derived class of your application, its pointer is retrieved and stored in **pApp**. **GetProfileString** takes two strings as parameters: first, the name of the section, and second, the name of the entry as used in **Gnd.ini**. The return value is the specific string referred to by those two parameters. The following declarations and definitions provide this functionality:

```
in Gnd.h:
struct PANSATFileInfo
{
  CString Dir;
  CString Ext;
  CString Des;
};

const static char *def[]  =    {"Script", "Macro", "Telemetry",
                                "Userlog", "Task", "In",
                                "Out"};
const static int MAXDIRS = sizeof(def)/sizeof(char *);

in Gndview.h:
  CString strSectionDir;
  CString strSectionExt;
  CString strSectionDscrpt;
  struct PANSATFileInfo PFI[MAXDIRS];
```

With the above code, the following Gnd.ini file can be read by the application:

```
[Extensions]
Script=*.PSF
Macro=*.GMD
Telemetry=*.TMY
Userlog=*.USL
Task=*.PTL
In=*.IN
Out=*.OUT

[Directories]
Script=D:\Ground\Script
Macro=D:\Ground\Macro
Telemetry=D:\Ground\Telmetry
Userlog=D:\Ground\Userlog
Task=D:\Ground\Task
In=D:\Ground\IN
Out=D:\Ground\OUT

[Descriptions]
Script=PANSAT Script File
Macro=Macro Definition
Telemetry=Telemetry Data
Userlog=User Log
Task=PANSAT Task List
In=IN Data
Out=OUT Data
```

Everything on the right side of the equal sign is stored into the appropriate members of the **PFI** structure. In case you want to store to instead of retrieve data from the Gnd.ini file, use code as shown below:

```
int i;
for (i=0; i<MAXDIRS; i++)
{
    pApp->WriteProfileString(strSectionDir, def[i], PFI[i].Dir);
}
```

This writes **MAXDIRS** entries from the **PFI[].Dir** string array after the equal sign following the string referred to in **def[]** in the **[Directories]** section of the Gnd.ini file. The sequence of strings in the Gnd.ini file does not necessarily have to be in the same order as the strings in the **PFI[].Dir**. The **XXXProfileString** functions will always resolve to the entry according the **string** defined by **def[]** (for this

example), not its index. This makes it even more convenient for the programmer to make use of *.ini files. With the MFC `CString` class both the use and the programming of *.ini files becomes a comparably easy task to accomplish.

4. MFC Class CString

The purpose of the MFC helper class `CString` is to simplify both memory allocation and string alteration. Refer to [Ref. 13] for a full description and explanation of the very useful methods of this class.

Why use a special class for strings when C/C++ already offers `char string[n]`'s? Because those variables are of fixed length, whereas `CString` instances may dynamically vary in length. This relieves the programmer from the repetitive and error prone task of checking for string boundaries, memory availability and dynamic memory allocation procedures and maintenance routines. With `CString`s, none of these tedious operations need to be coded by the programmer. Overridden operators such as =, +, += allow easy assignment and concatenation of strings to `CString` objects. Extraction and conversion routines make string alteration easy, as well as methods for comparing, searching and archiving strings.

The use of `CString` objects instead of `char` strings is strongly recommended. Try to replace every occurrence of the old-fashioned array of `char` by a `CString` object unless you definitely do not intend to change either the length or the contents of the string.

5. MFC Class CPtrArray and its Neighbors

In almost every application you will encounter the problem to store data structures and erase or edit some or all of them. Normally, it will not be known how many instances of a particular data structure are needed during development or compilation time. Memory as well as performance constraints thus will lead to a dynamic allocated and

released system of data structures. This will probably include linked lists and many pointers. However, this task can be left to one of the various MFC `CPtrArray` classes and its neighbors. Learn more about `CPtrArray`s from [Ref. 14]. Refer to a description for "Collection Classes" (a `CPtrArray`) in the same manual.

What kind of class is the best for your specific data structure storage problem? The answer to this question can only be found in your specific data structures and the needs imposed on your code to behave correctly. A short overview might be of assistance when browsing through the help files for Collection Classes. There are three basic ways of data storage supported by those classes; however, one thing is common to all of them: the programmer never has to worry about memory allocation or deallocation. The three different kinds of storage are:

- The `array`. It will behave like a zero-based C array, its members thus are accessable by their index. The array itself will grow or shrink as items are added or removed. The array can contain bytes, words, doublewords, strings, generic pointers and more variable types.

- The `list`. It will behave like a doubly-linked list in C. The list can contain strings, generic pointers, pointers to `CObject` classes and other variable types.

- The `map`. This involves variable types called *elements* which are *values* you attached a unique *key* to. Every value then is referenced by its key. Every map collection class is named `CMapXXXToYYY`, where `XXX` represents the key and `YYY` the value referenced by the key.

you The use of one or more of these collection classes is highly recommended because they will make programming of arrays, lists and mappable variables significantly easier. Spend one hour to get familiar with these, and save many more while programming them.

C.   BUG REPORT

This is a somewhat arbitrary collection of bugs, programming errors, runtime failures and other nasty problems that could happen to your application while developing.

They might considerably help locating bugs in a matter of minutes rather than days or even weeks. I would personally suggest that you write down the bugs YOU encountered for your personal records: including a short note, a description of the error and your workaround or whatever was helpful.

Nobody does it, everybody should: use *comments*! If you plan to develop an application for more than three weeks, you should comment your source files as precisely as you can. I can assure you would REALLY regret it after two months of development. Trust me!!

For further specific information on a topic check [Ref. 6] and browse through the headings.

1. Mysterious Syntax Errors While Using #define's

You might want to check if you terminated your `#define`'s with a semicolon; if so, delete the semicolon. Otherwise you will get error messages like

```
error C2143: syntax error : missing ')' before ';'
error C2059: syntax error : ')'
error C2181: illegal else without matching if
```

2. Access Violations Due to Bad Memory

This occurs most likely not because of defective RAM, but because you forgot to allocate memory for your object. This is not as obvious as it normally should be, especially when memory allocation is hidden somewhere in the framework.

Example: you created an AppWizard-application and tried to access `CDocument`-derived class members from within the `CView`-derived class constructor. This will fail somewhere in the framework source files with an Access Violation, failed **ASSERT**

statements or other error codes related to `CDocument`-derived class access, or, in the worst case, during runtime with a system crash.

The application does not run anymore because at the time the `CView`-derived class constructor is running, the `CDocument`-derived class is not yet created in memory. Thus, your code in the constructor relying on an existing `CDocument`-derived class just accesses random memory addresses, and fails somewhere when the randomly accessed memory contents does not make sense any more. This normally occurs quickly, but former `CDocument` instances could mislead your application, so you might encounter randomly running and crashing application behaviour.

This is a very ugly thing when it happens, so how do you find out about it beforehand? Normally, you will not be able to prevent yourself from programming such a nasty bug, unless you know exactly when classes are instantiated relative to others. This information is hidden sometimes in the **Overview** or **General Information** section in the online help for classes, but more often you will not find anything about it there. If so, look at the methods provided with these classes, and you will find one or more names indicating an initialization or presetting of member variables or class structures. Read the help text for those methods, and most likely you will find sentences like "This function is invoked after XYZ has been constructed, but just before ABC is shown.". The best place to look for such a method is to browse through the class "Initialization" and "Overridables" section. If this explanation sounds like a solution to your problem (remember, you do not really know why your application crashes all the time, you are just poking around suspiciously), just put those lines of your code which you expect to be the faulty ones (again, another thing to figure out) in an overridden instance of that specific method. This will clear everything after just minutes of work, if you guessed everything right. And, in the future you will be wiser.

3. AppWizard Does Not Recognize Your Classes

This is a problem that occurs sometimes when classes, which are not part of MFC or MFC derived classes, are used. In this case, delete the *.clw file from your development directory. All information that describes how ClassWizard needs to perform its actions is stored in this *.clw file. Before you can run ClassWizard again, the database must be reconstructed. This takes only a few seconds and could be achieved by following two steps: first, open the *.rc resource file from within the Visual Workbench. The MSVC built-in dialog editor AppWizard starts up and processes your resources. From the main menu, choose `Project/ClassWizard` and affirm the upcoming dialog box. Then a dialog box with the current project files comes up. You may specify additional source files whose contents you wish to be recognized by ClassWizard, but normally you just affirm the default settings. After that, the *.clw ClassWizard database is regenerated, and it now contains all class information including even the non-MFC classes and their derivatives.

Sometimes you might not be able to use the ClassWizard `Message Map` maintaining feature; ClassWizard does not offer `Add Function` for previously defined visual controls, even if you rebuilt the *.clw database file like described above. If so, you have to manually implement all the changes ClassWizard would have done for you automatically (described in "Using ClassWizard to Change Your Application").

4. Globally Defined Variables Are Not Recognized

You might encounter the problem that your globally defined variables are not recognized in classes you added to the project, even though you `#include`'d every necessary file into the class implementation, and neither the compiler nor the linker complained about undefined variables. This seems to be a compiler problem; however, I never experienced it before I started developing the groundstation application.

How do you learn about this error? The compiler and linker know about the global variable, but during runtime, just another address is resolved. This results in unexpected program behaviour of some kind related to that global variable - nothing new during the development phase. That this error exists can usually easily be discovered by invoking the QuickWatch feature while debugging the program (`Debug/QuickWatch` or Shift-F9 with cursor over the variable in question, or typing it in into the upcoming dialog box).

The solution is easy: define a member variable of the same type (or a pointer to it) in the class in which you want to use the inaccessible global variable, and set it to the value (or the address, in case of a pointer variable) of the global variable from within your `CView`- or `CFrameWnd`-derived class of the application.

VI. CONCLUSION

This thesis demonstrates that software development of a complex Windows-based application could be accomplished with satisfying results. For the implementation of the complete functionality of the PANSAT Software Groundstation, however, further development will be necessary. This thesis does not only discuss the Software Groundstation, but also provides the reader with the information necessary to develop other Windows-based C++ applications using MSVC and the various additional tools.

Software development is a delicate task. Poor conceptual design in the beginning can result in disastrous conditions during and after development. That is why most of the time and effort should be spent in defining the appropriate data structures and the conceptual design. Most often, errors because of poor conceptual design can only be corrected with a vast amount of time and manpower (that is, money), if at all. Thus, a certain portion of detail must be provided during this phase, otherwise the effort will be worthless.

This thesis also shows that software development will soon become a team effort, once the conceptual design reaches a certain complexity. Although only one person may actually code the program, he will be dependent on the technically funded inputs of his co-workers, as software usually puts a multitude of engineering tasks in one single environment.

VII. LIST OF REFERENCES

1. *WinWidgets/32 Programming Guide and Reference Manual*, Lifeboat Publishing, 1994

2. *Win32 Programmer's Reference*, Volume 1-2 Overview, Chapter 76: *Common Dialog Box Library*, MSVC Books Online

3. *Win32 Programmer's Reference*, Volume 1-2 Overview, Chapter 45: *Files*, MSVC Books Online

4. *Win32 Programmer's Reference*, Volume 1-2 Overview, Chapter 68: *Communications*, MSVC Books Online

5. *Win32 Programmer's Reference*, Volume 1-2 Overview, Chapter 44: *Synchronization*, MSVC Books Online

6. *MFC, MFC Technical Notes*, MSVC Books Online

7. *ODBC 2.0 SDK*, MSVC Books Online

8. *Introducing Visual C++*, Microsoft Corporation (installation manual shipped with the MSVC Compiler), 1994

9. *Tab Dialog Classes (MFC): CTabDlg and CTabDlgChild*, WinWidgets Manual - Tab Control - Tab helper classes

10. *User's Guides, Visual C++ User's Guide*, Part 1: *Using Visual C++*, Chapter 12: *Using ClassWizard*, MSVC Books Online

11. WinWidgets Manual, Online Manual

12. *MFC, Class Library Reference*, class CWinApp member functions, MSVC Books Online

13. *MFC, Class Library Reference*, class CString, MSVC Books Online

14. *MFC, Class Library Reference*, class CPtrArray, MSVC Books Online

15. Gregory Wade Lawrence, *Preliminary PANSAT Ground Station Software Design and Use of an Expert System to Analyze Telemetry*, Master's Thesis, Naval Postgraduate School, Monterey, March 1994

16. Troy M. Nichols, *A Description of the PANSAT Command Language*, Master's Thesis, Naval Postgraduate School, Monterey, September 1995

17. BekTek Spacecraft Operating System, *SCOS Reference Manual*, AMSAT-NA Microsat and UoSAT OBC186, December 1992

VIII. APPENDIX

A.   APPLICATION SOURCECODE

Gnd.h	Static variables/data structure (SCmd/SMacro) include file
Gnd.cpp	Application file implementation file
GndDoc.h	Groundstation Document include file
GndDoc.cpp	Groundstation Document and SCmd/SMacro implementation file
GndView.h	Groundstation View include file
GndView.cpp	Groundstation View/Message Map & Handler implementation file
MainFrm.h	Groundstation Main Frame include file
MainFrm.cpp	Groundstation Main Frame implementation file

Gnd.h

```
// Gnd.h : main header file for the GND application
//
// CAUTION: THIS CODE DEPENDS ON int BEING 4 BYTES LONG!!!

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"        // main symbols

/////////////////////////////////////////////////////
//
//  Common type definitions and static variables for the whole Gnd-project
//

struct PANSATFileInfo
{
  CString Dir;
  CString Ext;
  CString Des;
};
```

```cpp
struct SUser
{
  char login[20];
  char passw[12];
  int  flags;          // see Fxxx #define's below
};


// declaration of the SMacro structure
struct SMacro // A Macro definition. Every command can be regarded as a macro.
{
public:  // construction/destruction
  SMacro();
  SMacro(int nIndex);         // use for construction of built-in commands
  SMacro(const char *pName); // use for construction of macros (disk-stored commands)
  virtual ~SMacro();          // frees all allocated memory
public: // public member functions
  virtual int Load();         // overload this macro with a new one
  virtual int Save();  // save the current macro to disk
  virtual int Overwrite(); // overwrite the current macro to disk
  virtual int Execute(); // execute the macro
  virtual int GetError(); // retrieve nError if load on construction is used
  BOOL IsScript();
protected:  // protected member functions and member variables
  BOOL m_bHasChanged;
  int nError;  //internal error if Load, Save, Execute return FALSE; else amount of bytes read
private: // member functions unique to SMacro
  const char *GetFileName();  // use strcpy(char dest[256], pMacro->GetFileName());
  BOOL SetFileName(const char *pName);
  const char *GetMacroName();
  void SetMacroName(const char *pName);
private: // data members unique to SMacro
  CString FileName;
  CString MacroName;
  CPtrArray cmd;    // pointer to the macro CPtrArray structure containing SCmd structures
  BOOL m_bHasFileName;
  BOOL m_bIsBuiltIn;
  BOOL m_bIsScript;
// friend classes are...
  friend class CCHScriptsDlg;
};


#define NOERR                -1
#define ERR_FILE_NOT_FOUND    0
#define ERR_FILE_EXISTS       1
#define ERR_LOADING           2
#define ERR_SAVING            3
#define ERR_EXECUTING         4
```

```cpp
#define ERR_OVERWRITING            5
#define ERR_NO_FILENAME            6
#define ERR_OPEN_FOR_WRITING       7
#define ERR_OPEN_FOR_READING       8
#define ERR_OPEN_FOR_EXECUTING     9
#define ERR_WHILE_OBTAINING_FILESIZE 10
#define ERR_INVALID_FILENAME       11
#define ERR_TEMP_DELETION          12


const static char *ErrAry[] =
{
 "File not found!", //0
 "File already exists!", //1
 "Error while loading macro from disk!"//2
 "Error while saving macro to disk!", //3
 "Error while executing macro!", //4
 "Error while overwriting macro to disk!", //5
 "No file name specified!", //6
 "Couldn't open macro file for output!", //7
 "Couldn't open macro file for input!", //8
 "Couldn't open execution device!", //9
 "Error while trying to obtain the macro file size!", //10
 "This filename is invalid!", //11
 "Couldn't delete temporary file!" //12
};


struct SCmd : public SMacro // SCmd holds command and data info for output to PANSAT
{
public: // constructor/destructor
  SCmd();
  SCmd(int nIndex);
  ~SCmd();
public: // data members
  __int8  cmd;        // holds command ID according to DocCmd[].command
  __int16 wParam; // holds param of type specified by TW_xxx in DocCmd[].wParam_Type
  long    lParam; // holds param of type specified by TL_xxx in DocCmd[].lParam_Type
  void   *ptr;        // holds a generic pointer according to TQ_xxx-spec in DocCmd[].wParam_Type
public: // public member functions, overridden from SMacro
  virtual BOOL Load(void *fh);        // load a command from disk
  virtual BOOL Save(void *fh); // save stored command to disk
  virtual BOOL Execute(void *fh); // execute this command
protected:  // protected member functions and member variables
  virtual long GeneratePassword();
  BOOL ReadString(void *fh, long *pLong, int *pnErr);
  BOOL WriteString(void *fh, char *pchar, int *pnErr);
};
```

```cpp
struct SReturnCmd  // SReturnCmd holds command and data info for input from PANSAT
{
    __int8  cmd;       // holds command ID according to DocCmd[].command
    __int16 size;      // holds size of the whole data stream coming back or amount of received return structures
    void    *ptr;      // holds a generic pointer or other info according to TR_xxx-spec in DocCmd[].return_Type
};


// SDocumentCmd:
// The program command database. The source of information concerning command I/O with PANSAT.
struct SDocumentCmd  // holds internal command representation
{
    char   *command;     // plain command name
    __int8 cmdID;        // encoded command ID
    int    flags;        // Super User Command, Password, user accessability
    int    wParam_Type;  // defines type for SCmd.wParam
    int    lParam_Type;  // defines type for SCmd.lParam
    int    return_Type;  // defines return type for SReturnCmd.ptr
};


///////////////////////////////////////////////////////////
// parameter structures: Sxxx. Send this stuff UP TO PANSAT!!
//
struct STask
{
    char   *diskname; // filename of THIS task on disk; not to be sent to PANSAT
    char   *taskname; // taskname in PANSAT
    __int8 pri;       // task priority
    int    size;
    void   *data;     // task data has size bytes
public:
    STask();
    ~STask();
};


struct SOSParams
{
    // tbd
public:
    ~SOSParams();    // tbd
    SOSParams();     // tbd
};


////////////////////////////////////////////////////////////////////////////
// return structures: SRxxx.  CAUTION: Length of structures must be determinable!!!
//
struct SRFile  // holds one PANSAT file
{
```

```
  char *name;  // PANSAT file name has strlen(name) + 1 bytes
  void *data;  // file data has (SReturnCmd.size - strlen(name)+1) bytes
};


struct SRCmdBuffer : SCmd // holds one time tagged command (that is, command and
{                         // its parameters, planned begin of execution time)
  long  time; // SQLDATETIME4
};


struct SRTask  // holds info for one task in PANSAT
{
  char   *name;
  __int8  on;
  __int16 status;
  __int16 pri;
  __int16 size; // size in bytes of PANSAT task
};


struct SREvent : SRCmdBuffer // holds one event (that is, command and its parameters,
{                            // begin of execution time)
};


struct SRTelemetry
{
  // tbd
};


struct SROSParams
{
  int    version;
  long   time;         // SQLDATETIME4
  int    taskc;        // == n == # of entries taskv[n]. taskc=TaskCounter, taskv=TaskVector
  char **taskv;        // array of char * to tasknames
};


/////////////////////////////////////////////
// flags for SDocumentCmd and SUser: OR 'em!
#define FNOREQ 0x0000 // No requirements necessary
#define FPASS  0x0001 // Password required
#define FADMIN 0x0002 // must be administrator     to access super user and password cmds
#define FSUPER 0x0004 // must be super user        to access all Groundstation cmds
#define FINTRM 0x0008 // must be intermediate user to access all cmds except memory and low-level
#define FDUMB  0x0010 // stupid user - can only access mail cmds
#define FPCL   0x0020 // is a PCL command (if not set: is a Groundstation use command)


/////////////////////////////////////////////////////////////
// Types of parameters: SDocumentCmd.xx_Type identifier
```

```
//
// TW_xxx and TL_xxx define data types appearing in wParam and lParam of the SCmd structure respectively.
// TW_xxx used for both output and return to/from PANSAT, whereas TL_xxx is used only for output to PANSAT.
// TQ_xxx are OR'ed to TW_xxx and define how the SCmd.ptr member is to be interpreted. TQ_xxx are
// only used in the output structure SCmd.
// TR_xxx in contrary to TQ_xxx are only used in the return structure SReturnCmd. They're also OR'ed to
// TW_xxx to define the SReturnCmd.ptr generic pointer member.

#define TVOID      0x0000 // nothing specified
// wParam types: max#: 256, max range: 0x0000..0xFFFF
// Numbers
#define TW_AMOUNT 0x0001 // 0..65535 size of object
#define TW_DATA    0x0002 // 0..0xFF memory contents
#define TW_CMD     0x0003 // 0..255 PANSAT command ID

#define TW_MODE    0x0004 // byte length; 0x00=BPSK, 0xFF=SS
#define TW_CTRL    0x0005 // byte; bin xx000000..xx111111 (dec 63) transceiver control byte
#define TW_TABLE   0x0006 // tbd; so far 0..255
#define TW_LEVEL   0x0007 // tbd; so far 0..255
#define TW_POWER   0x0008

#define TW_VERSN   0x0009 // tbd; so far 0..255
#define TW_COUNT   0x000A // 0..200 amount of mem to be read fitting into one AX.25 frame

// Type qualifiers: TQ_xxx (max#: 256) to be OR'ed with one of the TW_yyy types
// Defines the contents of the SCmd.ptr
#define TQ_CMDPTR  0x0100 // SCmd.ptr contains a pointer to an SCmd structure
#define TQ_ADDRESS 0x0200 // SCmd.ptr contains a pointer to n bytes (n: to be OR'ed with this)
#define TQ_TASKPTR 0x0300 // SCmd.ptr contains a pointer to an STaskList structure
#define TQ_OSPTR   0x0400 // SCmd.ptr contains a pointer to an SOSParams structure

#define TQ_CHPTR   0x0500 // SCmd.ptr contains a pointer to a C string ('\0'-ended string)
#define TQ_MACRO   0x0600 // SCmd.ptr contains a pointer to an SMacro structure

// lParam types: max#: 65536; max range: 0x00000000..0xFFFFFFFF
// char *, Addresses, Time
#define TL_CHPTR   0x0001 // pointer to a C string
#define TL_PCBADR  0x0002 // 0..0xFF PCB address
#define TL_RAMADR  0x0003 // 0..7FFFF (dec 524287) and F0000..FFFFF (dec 983040..1048575) RAM/ROM address
#define TL_SRAMADR 0x0004 // 0..0x3FFFFF (dec 4194303) SRAM address
#define TL_FLSHADR 0x0005 // 0..0x7FFFF (dec 524287) flash mem address
#define TL_PORTADR 0x0006 // 0..0xFFFF (dec 65535) CPU I/O port address
#define TL_UTC     0x0007 // 4 byte UTC time/date, equivalent to 4 Byte SQL time/date

// SReturnCmd return type qualifiers: TR_xxx (max#: 256) to be OR'ed with one of the TW_yyy types
// Defines the contents of the SReturnCmd.ptr generic pointer
#define  TR_ADDRESS  0x0001 // address in groundstation computer memory
```

```c
#define   TR_UTC      0x0002 // SQLDATETIME4

#define   TR_CHPTR    0x0003 // pointer to C string
#define   TR_FILEPTR  0x0004 // pointer to struct SRFile
#define   TR_CMDPTR   0x0005 // pointer to struct SRCmdBuffer
#define   TR_TASKPTR  0x0006 // pointer to struct SRTask
#define   TR_EVENTPTR 0x0007 // pointer to struct SREvent
#define   TR_TELPTR   0x0009 // pointer to struct SRTelemetry
#define   TR_OSPTR    0x0008 // pointer to struct SROSParams

const static SDocumentCmd DocCmd[] =
{
  {"add_command"            , 0x01, FPCL|FSUPER|FPASS, TVOID    | TQ_CMDPTR , TL_UTC    , TVOID              },
  {"add_task"               , 0x02, FPCL|FSUPER|FPASS,             TQ_TASKPTR, TVOID     , TVOID             },
  {"boot_rom"               , 0x03, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"charge_batt_a"          , 0x04, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"charge_batt_b"          , 0x05, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"delete_command"         , 0x06, FPCL|FSUPER|FPASS, TW_CMD                , TL_UTC    , TVOID             },
  {"delete_file"            , 0x07, FPCL|FNOREQ      , TVOID                 , TL_CHPTR  , TVOID             },
  {"delete_task"            , 0x08, FPCL|FSUPER|FPASS, TVOID                 , TL_CHPTR  , TVOID             },
  {"directory"              , 0x09, FPCL|FNOREQ      , TVOID                 , TVOID     , TW_AMOUNT| TR_CHPTR },
  {"discharge_batt_a"       , 0x0A, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"discharge_batt_b"       , 0x0B, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"drop_users"             , 0x0C, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"get_file"               , 0x0D, FPCL|FNOREQ      , TVOID                 , TL_CHPTR  ,           TR_FILEPTR },
  {"io_read"                , 0x0E, FPCL|FNOREQ      , TVOID                 , TL_PORTADR, TW_DATA  | TR_ADDRESS },
  {"io_write"               , 0x0F, FPCL|FSUPER|FPASS, TW_DATA               , TL_PORTADR, TVOID             },
  {"list_command_buffer"    , 0x10, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TW_AMOUNT| TR_CMDPTR },
  {"list_tasks"             , 0x11, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TW_AMOUNT| TR_TASKPTR },
  {"lockout_users"          , 0x12, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"pcbr"                   , 0x13, FPCL|FNOREQ      , TVOID                 , TL_PCBADR , TW_DATA  | TR_ADDRESS },
  {"pcbw"                   , 0x14, FPCL|FSUPER|FPASS, TW_DATA               , TL_PCBADR , TVOID             },
  {"purge_command_buffer"   , 0x15, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"purge_event_log"        , 0x16, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"purge_flash_data"       , 0x17, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"purge_stored_telemetry" , 0x18, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TVOID             },
  {"put_file"               , 0x19, FPCL|FNOREQ      , TW_AMOUNT| TQ_ADDRESS, TL_CHPTR  , TVOID             },
  {"read_clock"             , 0x1A, FPCL|FNOREQ      , TVOID                 , TVOID     ,           TR_UTC   },
  {"read_event_log"         , 0x1B, FPCL|FSUPER|FPASS, TVOID                 , TVOID     , TW_AMOUNT| TR_EVENTPTR },
  {"read_flash_data"        , 0x1C, FPCL|FNOREQ      , TVOID                 , TVOID     ,           TR_TELPTR },
  {"read_flash_mem"         , 0x1D, FPCL|FNOREQ      , TW_AMOUNT             , TL_FLSHADR, TW_AMOUNT| TR_ADDRESS },
  {"read_os_param"          , 0x1E, FPCL|FNOREQ      , TVOID                 , TVOID     ,           TR_OSPTR  },
  {"read_recent_telemetry"  , 0x1F, FPCL|FNOREQ      , TVOID                 , TVOID     ,           TR_TELPTR },
  {"read_sram_mem"          , 0x20, FPCL|FNOREQ      , TW_AMOUNT             , TL_SRAMADR, TW_AMOUNT| TR_ADDRESS },
  {"read_stored_telemetry"  , 0x21, FPCL|FNOREQ      , TVOID                 , TVOID     ,           TR_TELPTR },
  {"reset_watchdog"         , 0x22, FPCL|FNOREQ      , TVOID                 , TVOID     , TVOID             },
  {"sel_trans_param"        , 0x23, FPCL|FNOREQ      , TW_CTRL               , TVOID     , TVOID             },
```

```
    {"select_xmit_mode"       , 0x24, FPCL|FSUPER|FPASS, TW_MODE               , TVOID      , TVOID                  },
    {"set_clock"              , 0x25, FPCL|FSUPER|FPASS, TVOID                 , TL_UTC     , TVOID                  },
    {"set_polling_rates"      , 0x26, FPCL|FSUPER|FPASS, TW_TABLE              , TVOID      , TVOID                  },
    {"set_power_level"        , 0x27, FPCL|FSUPER|FPASS, TW_LEVEL              , TVOID      , TVOID                  },
    {"set_pwr_switch"         , 0x28, FPCL|FSUPER|FPASS, TW_POWER              , TVOID      , TVOID                  },
    {"set_storage_rates"      , 0x29, FPCL|FSUPER|FPASS, TW_TABLE              , TVOID      , TVOID                  },
    {"start_task"             , 0x2A, FPCL|FSUPER|FPASS, TVOID                 , TL_CHPTR   , TVOID                  },
    {"stop_task"              , 0x2B, FPCL|FSUPER|FPASS, TVOID                 , TL_CHPTR   , TVOID                  },
    {"stop_wdog"              , 0x2C, FPCL|FSUPER|FPASS, TVOID                 , TVOID      , TVOID                  },
    {"unlock_users"           , 0x2D, FPCL|FSUPER|FPASS, TVOID                 , TVOID      , TVOID                  },
    {"update_os_param"        , 0x2E, FPCL|FSUPER|FPASS,             TQ_OSPTR  , TVOID      , TVOID                  },
    {"write_flash_mem"        , 0x2F, FPCL|FNOREQ      , TW_DATA               , TL_FLSHADR, TVOID                  },
    {"write_sram_mem"         , 0x30, FPCL|FSUPER|FPASS, TW_DATA               , TL_SRAMADR, TVOID                  }
/*
    {"read_memory"            , 0x31, FPCL|FNOREQ      , TW_AMOUNT             , TL_RAMADR , TW_AMOUNT|TR_ADDRESS  },
    {"write_memory"           , 0x32, FPCL|FNOREQ      , TW_AMOUNT| TQ_ADDRESS, TL_RAMADR , TVOID                  }
*/
};


const static int nAmountCmd = sizeof(DocCmd)/sizeof(SDocumentCmd);


// the following #define's allow symbolic access to all commands via their 0-based index entry into
// the main CPtrArray named cmds. DONT MESS WITH THESE DEFINES!
// THEY REPRESENT THE INDEX IN DocCmd[] AND ARE LATER STORED INTO CPtrArray Macro!


#define CMD_add_command           0x00
#define CMD_add_task              0x01
#define CMD_boot_rom              0x02
#define CMD_charge_batt_a         0x03
#define CMD_charge_batt_b         0x04
#define CMD_delete_command        0x05
#define CMD_delete_file           0x06
#define CMD_delete_task           0x07
#define CMD_directory             0x08
#define CMD_discharge_batt_a      0x09
#define CMD_discharge_batt_b      0x0A
#define CMD_drop_users            0x0B
#define CMD_get_file              0x0C
#define CMD_io_read               0x0D
#define CMD_io_write              0x0E
#define CMD_list_command_buffer   0x0F
#define CMD_list_tasks            0x10
#define CMD_lockout_users         0x11
#define CMD_pcbr                  0x12
#define CMD_pcbw                  0x13
#define CMD_purge_command_buffer  0x14
#define CMD_purge_event_log       0x15
```

```
#define CMD_purge_flash_data         0x16
#define CMD_purge_stored_telemetry   0x17
#define CMD_put_file                 0x18
#define CMD_read_clock               0x19
#define CMD_read_event_log           0x1A
#define CMD_read_flash_data          0x1B
#define CMD_read_flash_mem           0x1C
#define CMD_read_os_param            0x1D
#define CMD_read_recent_telemetry    0x1E
#define CMD_read_sram_mem            0x1F
#define CMD_read_stored_telemetry    0x20
#define CMD_reset_watchdog           0x21
#define CMD_sel_trans_param          0x22
#define CMD_select_xmit_mode         0x23
#define CMD_set_clock                0x24
#define CMD_set_polling_rates        0x25
#define CMD_set_power_level          0x26
#define CMD_set_pwr_switch           0x27
#define CMD_set_storage_rates        0x28
#define CMD_start_task               0x29
#define CMD_stop_task                0x2A
#define CMD_stop_wdog                0x2B
#define CMD_unlock_users             0x2C
#define CMD_update_os_param          0x2D
#define CMD_write_flash_mem          0x2E
#define CMD_write_sram_mem           0x2F
/*
#define CMD_read_memory              0x30
#define CMD_write_memory             0x31
*/
#define CMD_NEXT_VALUE                              0x30 // keep this one always the next free index!
#define CMD_macro01        CMD_NEXT_VALUE
#define CMD_macro02        CMD_NEXT_VALUE+1
#define CMD_macro03        CMD_NEXT_VALUE+2
#define CMD_macro04        CMD_NEXT_VALUE+3
#define CMD_macro05        CMD_NEXT_VALUE+4
#define CMD_macro06        CMD_NEXT_VALUE+5
#define CMD_macro07        CMD_NEXT_VALUE+6
#define CMD_macro08        CMD_NEXT_VALUE+7
#define CMD_macro09        CMD_NEXT_VALUE+8
#define CMD_macro10        CMD_NEXT_VALUE+9
#define CMD_macro11        CMD_NEXT_VALUE+10
#define CMD_macro12        CMD_NEXT_VALUE+11
#define CMD_macro13        CMD_NEXT_VALUE+12
#define CMD_macro14        CMD_NEXT_VALUE+13
#define CMD_macro15        CMD_NEXT_VALUE+14
```

```c
struct STypeRange
{
  int  typeID;      // type ID (TW_xxx and TL_xxx, see above #define's)
  int  vartype;     // variable type (Vxxx, see #define's below)
  long low;         // valid range for specified typeID:
  long high;        // - if vartype==VNUMBER, VADDRESS: low, high  : lowest/highest number,
  long xlow;        // xlow, xhigh: except, but not including, that range
  long xhigh;       // - if vartype==VSTRING: low: max string length (including \0)
                    // - if vartype==VTIME: no built-in restrictions
};
// variable types: STypeRange.vartype ....
#define VT_NUMBER  0x01
#define VT_ADDRESS 0x02
#define VT_STRING  0x03
#define VT_TIME    0x04

const static STypeRange TypeRange[] =
{
  { TW_AMOUNT , VT_NUMBER , 0, 0xFFFF, 0, 0   },
  { TW_DATA   , VT_NUMBER , 0, 0xFF  , 0, 0   },
  { TW_CMD    , VT_NUMBER , 0, 0xFF  , 0, 0   },

  { TW_MODE         , VT_NUMBER , 0, 0xFF  , 0, 0xFF},
  { TW_CTRL   , VT_NUMBER , 0, 0x3F  , 0, 0   },
  { TW_TABLE  , VT_NUMBER , 0, 0xFF  , 0, 0   },
  { TW_LEVEL  , VT_NUMBER , 0, 0xFF  , 0, 0   },
  { TW_POWER  , VT_NUMBER , 0, 0xFF  , 0, 0   },

  { TW_VERSN  , VT_NUMBER , 0, 0xFF  , 0, 0   },
  { TW_COUNT  , VT_NUMBER , 0, 0xC8  , 0, 0   },

  { TL_CHPTR  , VT_ADDRESS, 0, 0      , 0      , 0       },
  { TL_PCBADR , VT_ADDRESS, 0, 0xFF   , 0      , 0       },
  { TL_RAMADR , VT_ADDRESS, 0, 0xFFFFF , 0x7FFFF, 0xF0000},
  { TL_SRAMADR, VT_ADDRESS, 0, 0x3FFFFF, 0      , 0       },
  { TL_FLSHADR, VT_ADDRESS, 0, 0x7FFFF, 0      , 0       },
  { TL_PORTADR, VT_ADDRESS, 0, 0xFFFF , 0      , 0       },
  { TL_UTC    , VT_TIME   , 0, 0      , 0      , 0       },
};


// default sections in the Gnd.INI file:
const static char *def[]  = {"Script", "Macro", "Telemetry", "Userlog", "Task", "In", "Out"};
const static int MAXDIRS = sizeof(def)/sizeof(char *);
#define IX_SCRIPT    0
#define IX_MACRO     1
#define IX_TELEMETRY 2
#define IX_USERLOG   3
```

```
#define IX_TASK      4
#define IX_IN        5
#define IX_OUT       6
// edit IDs in Preferences-Dialog (see GND.RC):
const static int prefID[] = { IDC_EDIT1, IDC_EDIT2, IDC_EDIT3, IDC_EDIT4,
                              IDC_EDIT5, IDC_EDIT6, IDC_EDIT7 };


/////////////////////////////////////////////////////////////////////////
// CGndApp:
// See Gnd.cpp for the implementation of this class
//

class CGndApp : public CWinApp
{
public:
        CGndApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CGndApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(CGndApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

Gnd.cpp

```
// Gnd.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Gnd.h"

#include "mainfrm.h"
#include "Gnddoc.h"
#include "mytabdlg.h"
#include "Gndview.h"
```

```
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////////
// CGndApp


BEGIN_MESSAGE_MAP(CGndApp, CWinApp)
        //{{AFX_MSG_MAP(CGndApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CGndApp construction


CGndApp::CGndApp()
{
  m_pszAppName="PANSAT Ground Station";
  m_pszProfileName="Gnd.INI";
}


/////////////////////////////////////////////////////////////////////////////
// The one and only CGndApp object


CGndApp theApp;


/////////////////////////////////////////////////////////////////////////////
// CGndApp initialization


BOOL CGndApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

        Enable3dControls();
```

```
        LoadStdProfileSettings();  // Load standard INI file options (including MRU)
        WidgetsInit();
        //XTableInit();
        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.

        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CGndDoc),
                RUNTIME_CLASS(CMainFrame),          // main SDI frame window
                RUNTIME_CLASS(CGndView));
        AddDocTemplate(pDocTemplate);

        // create a new (empty) document
        OnFileNew();

        if (m_lpCmdLine[0] != '\0')
        {
                // TODO: add command line processing here
        }

        return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //{{AFX_MSG(CAboutDlg)
                // No message handlers
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

86

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}


void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


// App command to run the dialog
void CGndApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}


/////////////////////////////////////////////////////////////////////////////
// CGndApp commands
```

GndDoc.h

```
// Gnddoc.h : interface of the CGndDoc class
//
/////////////////////////////////////////////////////////////////////////////


class CGndDoc : public CDocument
{
protected: // create from serialization only
        CGndDoc();
        DECLARE_DYNCREATE(CGndDoc)

// Attributes
public:
  SMacro *pMacro;
  CPtrArray c; // home of all built-in commands
```

```
  CPtrArray m; // home of all loaded macros
// Operations
public:


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CGndDoc)
        public:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL


// Implementation
public:
        virtual ~CGndDoc();
        virtual void Serialize(CArchive& ar);    // overridden for document i/o
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:


// Generated message map functions
protected:
        //{{AFX_MSG(CGndDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
```

GndDoc.cpp

```
// Gnddoc.cpp : implementation of the CGndDoc class
//

#include "stdafx.h"
#include "fcntl.h"
#include "sys/stat.h"
#include "Gnd.h"

#include "Gnddoc.h"
                          .

#ifdef _DEBUG
```

```
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////
// CGndDoc

IMPLEMENT_DYNCREATE(CGndDoc, CDocument)

BEGIN_MESSAGE_MAP(CGndDoc, CDocument)
        //{{AFX_MSG_MAP(CGndDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //     DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
// CGndDoc construction/destruction

CGndDoc::CGndDoc()
{
  //pMacro = new SMacro("D:\\ich\\hier\\kaum\\daten.PCL");
}

CGndDoc::~CGndDoc()
{
 // delete pMacro;
}

BOOL CGndDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////
// CGndDoc serialization

void CGndDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
```

89

```
                        // TODO: add storing code here
            }
            else
            {
                        // TODO: add loading code here
            }
}


/////////////////////////////////////////////////////////////////////////////
// CGndDoc diagnostics

#ifdef _DEBUG
void CGndDoc::AssertValid() const
{
            CDocument::AssertValid();
}

void CGndDoc::Dump(CDumpContext& dc) const
{
            CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////
// SCmd implementation
//
SCmd::SCmd()
{
  cmd    = 0;
  wParam = 0;
  lParam = 0;
  ptr    = (void *)0;
}

SCmd::SCmd(int nIndex)
{
  cmd    = nIndex;
  wParam = 0;
  lParam = 0;
  ptr    = (void *)0;
}

SCmd::~SCmd()
{
  int n;

  for (n=0; n<nAmountCmd, DocCmd[n].cmdID!=cmd; n++);
```

```
        switch (DocCmd[n].wParam_Type & 0xFF00)
        {
                case TQ_CMDPTR : if ((SCmd *)ptr)       delete (SCmd *)ptr;       break;
                case TQ_TASKPTR: if ((STask *)ptr)      delete (STask *)ptr;      break;
                case TQ_OSPTR  : if ((SOSParams *)ptr)  delete (SOSParams *)ptr;  break;
        }
        switch (DocCmd[n].lParam_Type)
        {
          case TL_CHPTR  : if ((SCmd *)lParam)    delete (SCmd *)lParam;    break;
        }
}


BOOL SCmd::ReadString(void *fh, long *pLong, int *pnErr)
{
  BOOL err;
  char c;
  CString str = "";

  for (; err = ReadFile(fh, &c, 1, (LPDWORD)pnErr, NULL), *pnErr!=0, c!='\0'; str += c);
  if (!(*pnErr) || err) return FALSE; // unexpected eof
  else
  {
    str += '\0';
    *pLong = (long)(new char[str.GetLength()+1]);
    strcpy((char *)(*pLong), LPCTSTR(str));
    return TRUE;
  }
}


BOOL SCmd::Load(void *fh) // load commands as they were saved (see SCmd::Save): only commands
{                         // without password and referenced files (normally from disk)
  int  n, wType, qType, lType;
  int th;          // temp handle for add_task, put_file commands
  DWORD tlen;      // temp file length

  if (ReadFile(fh, &cmd, 1, (LPDWORD)&nError, NULL) && nError==0) return TRUE; // eof
  for (n=0; n<nAmountCmd, DocCmd[n].cmdID!=cmd; n++);

  wType = DocCmd[n].wParam_Type & 0x00FF;
  qType = DocCmd[n].wParam_Type & 0xFF00;
  lType = DocCmd[n].lParam_Type;

  switch(wType)
  {
    case TVOID    : switch (lType)
                    {
                      case TL_PORTADR:
```

91

```
                      case TL_PCBADR :
                      case TL_FLSHADR:
                      case TL_SRAMADR:
                      case TL_UTC    : if (!ReadFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                      case TL_CHPTR  : if (!ReadString(fh, &lParam, &nError)) return FALSE;
                                         else break; // go on with (a)
                  }
                  switch (qType) // (a):
                  {
                    case TQ_CMDPTR : ptr = new SCmd;
                                       if (!(( (SCmd *)(ptr) )->Load(fh))) return FALSE; else break;
                    case TQ_TASKPTR: ptr = new STask;
                                       if (!ReadString(fh, (long *)(&( (STask *)(ptr) )->taskname), &nError))
                                         return FALSE;
                                       if (!ReadString(fh, (long *)(&( (STask *)(ptr) )->diskname), &nError))
                                         return FALSE;
                                       if (!ReadFile(fh, &( (STask *)(ptr) )->pri, 1, (LPDWORD)&nError, NULL))
                                         return FALSE;
                                       if (th = CreateFile(( (STask *)(ptr) )->diskname, GENERIC_READ,
FILE_SHARE_READ,
                                         NULL, OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL)
                                            == INVALID_HANDLE_VALUE) return FALSE;
                                       else
                                       {
                                         if (tlen = GetFileSize((void *)th, NULL) == INVALID_FILE_SIZE ||
                                           tlen > 0xFFFF)
                                         {
                                           CloseHandle((void *)th);
                                           return FALSE;
                                         }
                                         else
                                         {
                                           ( (STask *)(ptr) )->size = tlen;
                                           ( (STask *)(ptr) )->data = new char[tlen];
                                           if (!ReadFile((void *)th, &( (STask *)(ptr) )->data, tlen,
                                           (LPDWORD)&nError, NULL)) return FALSE;
                                           CloseHandle((void *)th);
                                         }
                                       }
                                       break; // go on with (b)
                    case TQ_OSPTR  : break; // tbd
                  }
                  break; // (b): go on with (z)

case TW_CMD    : if (!ReadFile(fh, &wParam, 1, (LPDWORD)&nError, NULL)) return FALSE;
                 switch (lType)
                 {
```

```
                case TL_UTC    : if (!ReadFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                                 else break; // go on with (c)
            }
            break; // (c): go on with (z)


case TW_DATA  : switch (lType)
            {
                case TL_PORTADR:
                case TL_PCBADR :
                case TL_FLSHADR:
                case TL_SRAMADR: if (!ReadFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                                 if (!ReadFile(fh, &wParam, 1, (LPDWORD)&nError, NULL)) return FALSE;
                                 else break; // go on with (d)
            }
            break; // (d): go on with (z)


case TW_AMOUNT: switch (lType)
            {
                case TL_FLSHADR:
                case TL_SRAMADR:
                case TL_RAMADR : if (!ReadFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                                 if (!ReadFile(fh, &wParam, 2, (LPDWORD)&nError, NULL)) return FALSE;
                                 else break; // go on with (e)
                case TL_CHPTR  : if (qType==TQ_ADDRESS) // load file to mem (i.e. put_file)
                                 {
                                   if (!ReadString(fh, &lParam, &nError)) return FALSE;
                                   if (th = CreateFile((char *)lParam, GENERIC_READ, FILE_SHARE_READ, NULL,
                                                 OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL)
                                        == INVALID_HANDLE_VALUE) return FALSE;
                                   else
                                   {
                                     if (tlen = GetFileSize((void *)th, NULL) == INVALID_FILE_SIZE ||
                                         tlen > 0xFFFF)
                                     {
                                       CloseHandle((void *)th);
                                       return FALSE;
                                     }
                                     else
                                     {
                                       wParam = (__int16)tlen;
                                       ptr    = new char[tlen];
                                       if (!ReadFile((void *)th, ptr, tlen, (LPDWORD)&nError, NULL)) return
FALSE;
                                       CloseHandle((void *)th);
                                     }
                                   }
                                 }
```

```
                                      break; // go on with (e)
                          }
                          break; // (e): go on with (z)

        case TW_CTRL  :
        case TW_MODE  :
        case TW_TABLE :
        case TW_LEVEL :
        case TW_POWER : if (!ReadFile(fh, &wParam, 1, (LPDWORD)&nError, NULL)) return FALSE;
  }


  return TRUE; // (z):
}


BOOL SCmd::WriteString(void *fh, char *pchar, int *pnErr)
{
  char *p;
  if (!pchar) return FALSE;
  for (p=pchar; WriteFile(fh, p, 1, (LPDWORD)pnErr, NULL); p++)
    if (*p=='\0') return TRUE;
  return FALSE;
}


BOOL SCmd::Save(void *fh) // put out commands without password and without referenced files (normally to disk)
{
  int  n, wType, qType, lType;

  if (!WriteFile(fh, &cmd, 1, (LPDWORD)&nError, NULL)) return FALSE;
  for (n=0; n<nAmountCmd, DocCmd[n].cmdID!=cmd; n++);

  wType = DocCmd[n].wParam_Type & 0x00FF;
  qType = DocCmd[n].wParam_Type & 0xFF00;
  lType = DocCmd[n].lParam_Type;

  switch(wType)
  {
      case TVOID    : switch (lType)
                      {
                        case TL_PORTADR:
                        case TL_PCBADR :
                        case TL_FLSHADR:
                        case TL_SRAMADR:
                        case TL_UTC    : if (!WriteFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                        case TL_CHPTR  : if (!WriteString(fh, (char *)&lParam, &nError)) return FALSE;
                                         else break; // go on with (a)
                      }
                      switch (qType) // (a):
                      {
```

```c
                            case TQ_CMDPTR : if (!(( (SCmd *)(ptr) )->Save(fh))) return FALSE; else break;
                            case TQ_TASKPTR: if (!WriteString(fh, (char *)(&( (STask *)(ptr) )->taskname), &nError))
                                                return FALSE;
                                             if (!WriteString(fh, (char *)(&( (STask *)(ptr) )->diskname), &nError))
                                                return FALSE;
                                             if (!WriteFile(fh, &( (STask *)(ptr) )->pri, 1, (LPDWORD)&nError, NULL))
                                                return FALSE;
                                             break; // go on with (b)
                            case TQ_OSPTR  : break; // tbd
                          }
                        break; // (b): go on with (z)


        case TW_CMD   : if (!WriteFile(fh, &wParam, 1, (LPDWORD)&nError, NULL)) return FALSE;
                        switch (lType)
                        {
                          case TL_UTC    : if (!WriteFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                                           else break; // go on with (c)
                        }
                        break; // (c): go on with (z)


        case TW_DATA  : switch (lType)
                        {
                          case TL_PORTADR:
                          case TL_PCBADR :
                          case TL_FLSHADR:
                          case TL_SRAMADR: if (!WriteFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                                           if (!WriteFile(fh, &wParam, 1, (LPDWORD)&nError, NULL)) return FALSE;
                                           else break; // go on with (d)
                        }
                        break; // (d): go on with (z)


        case TW_AMOUNT: switch (lType)
                        {
                          case TL_FLSHADR:
                          case TL_SRAMADR:
                          case TL_RAMADR : if (!WriteFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                                           if (!WriteFile(fh, &wParam, 2, (LPDWORD)&nError, NULL)) return FALSE;
                                           else break; // go on with (e)
                          case TL_CHPTR  : if (qType==TQ_ADDRESS)
                                              if (!WriteString(fh, (char *)&lParam, &nError)) return FALSE;
                                           break; // go on with (e)
                        }
                        break; // (e): go on with (z)
        case TW_CTRL  :
        case TW_MODE  :
        case TW_TABLE :
        case TW_LEVEL :
```

```
          case TW_POWER : if (!WriteFile(fh, &wParam, 1, (LPDWORD)&nError, NULL)) return FALSE;
  }


  return TRUE; // (z):
}


BOOL SCmd::Execute(void *fh) // put out commands, their password and referenced files (normally to COM1)
{
  int  n, wType, qType, lType;
  int th; // temp handle for add_task, put_file commands
  DWORD tlen; // temp file length
  long lPassWord;

  if (!WriteFile(fh, &cmd, 1, (LPDWORD)&nError, NULL)) return FALSE;
  for (n=0; n<nAmountCmd, DocCmd[n].cmdID!=cmd; n++);
  if (DocCmd[n].flags & FPASS)
  {
    lPassWord = GeneratePassword();
    if (!WriteFile(fh, &lPassWord, 4, (LPDWORD)&nError, NULL)) return FALSE;
  }

  wType = DocCmd[n].wParam_Type & 0x00FF;
  qType = DocCmd[n].wParam_Type & 0xFF00;
  lType = DocCmd[n].lParam_Type;

  switch(wType)
  {
    case TVOID    : switch (lType)
                    {
                      case TL_PORTADR:
                      case TL_PCBADR :
                      case TL_FLSHADR:
                      case TL_SRAMADR:
                      case TL_UTC    : if (!WriteFile(fh, &lParam, 4, (LPDWORD)&nError, NULL}) return FALSE;
                      case TL_CHPTR  : if (!WriteString(fh, (char *)&lParam, &nError)) return FALSE;
                                       else break; // go on with (a)
                    }
                    switch (qType) // (a):
                    {
                    case TQ_CMDPTR : if (!(( (SCmd *)(ptr) )->Execute(fh))) return FALSE; else break;
                    case TQ_TASKPTR: if (!WriteString(fh, ( (STask *)(ptr) )->taskname, &nError)) return FALSE;
                                     if (!WriteFile(fh, &( (STask *)(ptr) )->pri, 1, (LPDWORD)&nError, NULL))
                                      return FALSE;
                                     if (th = CreateFile(( (STask *)(ptr) )->diskname, GENERIC_READ,
FILE_SHARE_READ,
                                         NULL, OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL)
                                             == INVALID_HANDLE_VALUE) return FALSE;
```

```
                          else
                          {
                            if (tlen = GetFileSize((void *)th, NULL) == INVALID_FILE_SIZE ||
                               tlen > 0xFFFF)
                            {
                              CloseHandle((void *)th);
                              return FALSE;
                            }
                            else
                            {
                              ( (STask *)(ptr) )->size = tlen;
                              ( (STask *)(ptr) )->data = new char[tlen];
                              if (!ReadFile((void *)th, &( (STask *)(ptr) )->data, tlen,
(LPDWORD)&nError,

                                  NULL)) return FALSE;
                              CloseHandle((void *)th);
                            }
                          }
                          if (!WriteFile(fh, &( (STask *)(ptr) )->size, 2, (LPDWORD)&nError, NULL))
                           return FALSE;
                          if (!WriteFile(fh, &( (STask *)(ptr) )->data, tlen, (LPDWORD)&nError, NULL))
                           return FALSE;
                          break; // go on with (b)
                  case TQ_OSPTR  : break; // tbd
                }
                break; // (b): go on with (z)

    case TW_CMD   : if (!WriteFile(fh, &wParam, 1, (LPDWORD)&nError, NULL)) return FALSE;
                  switch (lType)
                  {
                    case TL_UTC   : if (!WriteFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                               else break; // go on with (c)
                  }
                  break; // (c): go on with (z)

    case TW_DATA  : switch (lType)
                  {
                    case TL_PORTADR:
                    case TL_PCBADR :
                    case TL_FLSHADR:
                    case TL_SRAMADR: if (!WriteFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                                 if (!WriteFile(fh, &wParam, 1, (LPDWORD)&nError, NULL)) return FALSE;
                                 else break; // go on with (d)
                  }
                  break; // (d): go on with (z)

    case TW_AMOUNT: switch (lType)
```

```
                    {
                case TL_FLSHADR:
                case TL_SRAMADR:
                case TL_RAMADR : if (!WriteFile(fh, &lParam, 4, (LPDWORD)&nError, NULL)) return FALSE;
                                 if (!WriteFile(fh, &wParam, 2, (LPDWORD)&nError, NULL)) return FALSE;
                                 else break; // go on with (e)
                case TL_CHPTR  : if (qType==TQ_ADDRESS) // load file to mem (i.e. put_file)
                                 {
                                     if (!WriteString(fh, (char *)&lParam, &nError)) return FALSE;
                                     if (th = CreateFile((char *)lParam, GENERIC_READ, FILE_SHARE_READ, NULL,
                                                   OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL)
                                            == INVALID_HANDLE_VALUE) return FALSE;
                                     else
                                     {
                                       if (tlen = GetFileSize((void *)th, NULL) == INVALID_FILE_SIZE ||
                                           tlen > 0xFFFF)
                                       {
                                         CloseHandle((void *)th);
                                         return FALSE;
                                       }
                                       else
                                       {
                                         wParam = (__int16)tlen;
                                         ptr    = new char[tlen];
                                         if (!ReadFile((void *)th, ptr, tlen, (LPDWORD)&nError, NULL))
                                             return FALSE;
                                         CloseHandle((void *)th);
                                       }
                                     }
                                     if (!WriteFile(fh, &( (STask *)(ptr) )->size, 2, (LPDWORD)&nError, NULL))
                                             return FALSE;
                                     if (!WriteFile(fh, &( (STask *)(ptr) )->data, tlen, (LPDWORD)&nError, NULL))
                                             return FALSE;
                                     break; // go on with (e)
                                 }
                                 break; // (e): go on with (z)
                case TW_CTRL  :
                case TW_MODE  :
                case TW_TABLE :
                case TW_LEVEL :
                case TW_POWER : if (!WriteFile(fh, &wParam, 1, (LPDWORD)&nError, NULL)) return FALSE;
  }
  return TRUE; // (z):
}


long SCmd::GeneratePassword()
```

98

```
{
  return 0xABCD;
}


////////////////////////////////////////////////////////////////////////////
// SMacro implementation

SMacro::SMacro()
{
  nError = NOERR;
  m_bHasFileName = FALSE;
  m_bHasChanged  = FALSE;
  m_bIsBuiltIn   = TRUE;
  m_bIsScript    = TRUE;
  FileName  = "";
  MacroName = "";
}


SMacro::SMacro(int nIndex)
{
  nError = NOERR;
  m_bHasFileName = FALSE;
  m_bHasChanged  = FALSE;
  m_bIsBuiltIn   = TRUE;
  m_bIsScript    = TRUE;
  FileName  = "";

  MacroName = DocCmd[nIndex].command;
  cmd.Add(new SCmd(nIndex)); // call constructor of SCmd
}


SMacro::SMacro(const char *pName)
{
  int ixl, ixr, d;

  nError = NOERR;
  m_bHasFileName = TRUE;
  m_bHasChanged  = FALSE;
  m_bIsBuiltIn   = FALSE;
  FileName = pName;
  ixr = FileName.ReverseFind('.');
  ixl = FileName.ReverseFind(':')+1;
  if ((d=FileName.ReverseFind('\\')+1) > ixl) ixl = d;
  if (ixr==-1) MacroName = FileName.Mid(ixl);
  else MacroName = FileName.Mid(ixl, ixr-ixl);

  nError = Load();
```

```
}

SMacro::~SMacro()
{
  int i, imax;
  SCmd *pCmd;

  for (i=0, imax=cmd.GetUpperBound(); i<=imax; i++)
  {
    pCmd = (SCmd *)cmd.GetAt(i);
    delete pCmd;
  }
}


int SMacro::Load()
{
  int fh;
  DWORD flen;
  SCmd *pCmd;

  if (LPCTSTR(FileName)=="")
    return ERR_NO_FILENAME;

  if (fh = CreateFile(LPCTSTR(FileName), GENERIC_READ, FILE_SHARE_READ, NULL,
                        OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL)
          == INVALID_HANDLE_VALUE)
  {
    switch (GetLastError())
    {
      case ERROR_FILE_NOT_FOUND : return ERR_FILE_NOT_FOUND;
      default                   : return ERR_OPEN_FOR_READING;
    }
  }

  if (flen = GetFileSize((void *)fh, NULL) == INVALID_FILE_SIZE)
  {
    CloseHandle((void *)fh);
    return ERR_WHILE_OBTAINING_FILESIZE;
  }

  do
  {
    cmd.Add(pCmd = new SCmd());
    if (!pCmd->Load((void *)fh))
    {
      CloseHandle((void *)fh);
      return ERR_LOADING;
```

```
      }
    }
    while (pCmd->nError != 0); // Load ok, so nError contains amount of bytes read

    CloseHandle((void *)fh);
    IsScript();
    return NOERR;
}


BOOL SMacro::IsScript()
{
    int i, imax, n;

    for (i=0, imax=cmd.GetUpperBound(); i<=imax; i++)
    {
        for (n=0; n<nAmountCmd, DocCmd[n].cmdID!=((SCmd *)(cmd[i]))->cmd; n++);
        if (!(m_bIsScript = DocCmd[n].flags & FPCL)) break;
    }
    return m_bIsScript;
}


int SMacro::Save()
{
    int fh;
    int i, imax;

    if (!m_bHasFileName)
        return ERR_NO_FILENAME;

    if (fh = CreateFile(LPCTSTR(FileName), GENERIC_WRITE, 0, NULL,
                        CREATE_NEW, FILE_FLAG_WRITE_THROUGH, NULL)
            == INVALID_HANDLE_VALUE)
    {
        switch (GetLastError())
        {
            case ERROR_FILE_EXISTS : return ERR_FILE_EXISTS;
            default                : return ERR_OPEN_FOR_WRITING;
        }
    }

    for (i=0, imax=cmd.GetUpperBound(); i<=imax; i++)
        if (! ( ((SCmd *)(cmd.GetAt(i)))->Save((void *)fh)) )
        {
            CloseHandle((void *)fh);
            return ERR_SAVING;
        }
```

```
    CloseHandle((void *)fh);
    return NOERR;
  }


int SMacro::Overwrite()
{
  int fh;
  int i, imax;

  if (!m_bHasFileName)
    return ERR_NO_FILENAME;

  if (fh = CreateFile(LPCTSTR(FileName), GENERIC_WRITE, 0, NULL,
                      CREATE_ALWAYS, FILE_FLAG_WRITE_THROUGH, NULL)
          == INVALID_HANDLE_VALUE)
    return ERR_OPEN_FOR_WRITING;


  for (i=0, imax=cmd.GetUpperBound(); i<=imax; i++)
    if (! ( ((SCmd *)(cmd.GetAt(i)))->Save((void *)fh)) )
    {
      CloseHandle((void *)fh);
      return ERR_OVERWRITING;
    }

  CloseHandle((void *)fh);
  return NOERR;
}


int SMacro::Execute()
{
  int fh;
  int i, imax;

  if (!m_bHasFileName)
    return ERR_NO_FILENAME;

  if (fh = CreateFile("COM1", GENERIC_WRITE, 0, NULL,
                      OPEN_EXISTING, FILE_FLAG_WRITE_THROUGH, NULL)
          == INVALID_HANDLE_VALUE)
    return ERR_OPEN_FOR_EXECUTING;

  for (i=0, imax=cmd.GetUpperBound(); i<=imax; i++)
    if (! ( ((SCmd *)(cmd.GetAt(i)))->Execute((void *)fh)) )
    {
      CloseHandle((void *)fh);
      return ERR_EXECUTING;
    }
```

102

```
    CloseHandle((void *)fh);
    return NOERR;
}


int SMacro::GetError()
{
    return nError;
}


const char *SMacro::GetFileName()
{
    if (m_bHasFileName)
        return LPCTSTR(FileName);
    else return NULL;
}


BOOL SMacro::SetFileName(const char *pName)
{
    int fh;

    if (fh = CreateFile(pName, GENERIC_WRITE, 0, NULL,
                        CREATE_NEW, FILE_ATTRIBUTE_NORMAL, NULL)
            == INVALID_HANDLE_VALUE)
    {
        if (GetLastError() == ERROR_FILE_EXISTS)
        {
            FileName = pName; // file exists -> name ok
            return TRUE;
        }
        else
        {
            nError = ERR_INVALID_FILENAME;
            return FALSE; // invalid handle AND file doesn't exist -> invalid name
        }
    }
    else // valid handle -> file did not exist & was created -> name ok
    {
        CloseHandle((void *)fh); // close file before deletion!!
        if (!DeleteFile(pName))
        {
            nError = ERR_TEMP_DELETION; // THIS should never happen!
            return FALSE; // valid handle AND file couldn't be deleted -> ???
        }
        FileName = pName;
        return TRUE;
    }
```

```
}

const char *SMacro::GetMacroName()
{
  return LPCTSTR(MacroName);
}

void SMacro::SetMacroName(const char *pName)
{
  // TOWORK: check for validity of pName as a macroname
  MacroName = pName;
}

//////////////////////////////////////
// STask constructor/destructor
//
STask::STask()
{
  diskname=taskname=(char *)0;
  pri=0;
  size=0;
  data=NULL;
}

STask::~STask()
{
  if (diskname) delete diskname;
  if (taskname)    delete taskname;
  if (data)                delete data;
}

//////////////////////////////////////
// SOSParams constructor/destructor
//
SOSParams::SOSParams()
{
  // tbd
}

SOSParams::~SOSParams()
{
  // tbd
}

/////////////////////////////////////////////////////////////////////////
// CGndDoc commands
```

104

GndView.h

```
// Gndview.h : interface of the CGndView class
//
/////////////////////////////////////////////////////////////////////////


class CGndView : public CView
{
protected: // create from serialization only
        CGndView();
        DECLARE_DYNCREATE(CGndView)

// Attributes
public:
        CGndDoc* GetDocument();

// Operations
public:
  CMainTabDlg      *m_pTabDlg;  // main dialog holding all childs (from CTabDlg)

  CCHScriptsDlg     m_ChDlg0;         // Child tab dialogs (from CTabDlgChild)
  CCHTelemetryDlg   m_ChDlg1;         // telemetry data display
  CCHMailDlg        m_ChDlg2;    // non-user mailing surveillance/maintenance
  CCHMemoryDlg      m_ChDlg3;         // memory peek/poke
  CCHControlDlg     m_ChDlg4;         // low-level SCOS and PANSAT functions
  CCHOSControlDlg   m_ChDlg5;         // high-level functions: T/T cmd., Event Log, User Control
  CCHFileSystemDlg  m_ChDlg6;         // file system maintenance
  CCHTaskControlDlg m_ChDlg7;         // task control & maintenance
  struct PANSATFileInfo PFI[MAXDIRS];


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CGndView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual void OnInitialUpdate();
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL


// Implementation
public:
        virtual ~CGndView();
```

```
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif


protected:
// Generated message map functions
  BOOL m_bTabDlgUp;
  CString strSectionDir;
  CString strSectionExt;
  CString strSectionDscrpt;
  CString strSectionMacro;


protected:
        //{{AFX_MSG(CGndView)
        afx_msg void OnUserAccess();
        afx_msg void OnEndUserAccess();
        afx_msg void OnPreferences();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


#ifndef _DEBUG  // debug version in Gndview.cpp
inline CGndDoc* CGndView::GetDocument()
   { return (CGndDoc*)m_pDocument; }
#endif


/////////////////////////////////////////////////////////////////////////////
```

GndView.cpp

```
// Gndview.cpp : implementation of the CGndView class
//

#include "stdafx.h"
#include "Gnd.h"

#include "Gnddoc.h"
#include "mytabdlg.h"
#include "password.h"
#include "prefdlg.h"
#include "Gndview.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
```

```
#endif

//////////////////////////////////////////////////////////////////////////
// CGndView

IMPLEMENT_DYNCREATE(CGndView, CView)

BEGIN_MESSAGE_MAP(CGndView, CView)
        //{{AFX_MSG_MAP(CGndView)
        ON_COMMAND(ID_ACCESS_LOGON, OnUserAccess)
        ON_COMMAND(ID_ACCESS_LOGOFF, OnEndUserAccess)
        ON_COMMAND(ID_PREFERENCES, OnPreferences)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
// CGndView construction/destruction

CGndView::CGndView()
{
  int i;
  strSectionDir    = "Directories";
  strSectionExt    = "Extensions";
  strSectionDscrpt = "Descriptions";

  CGndApp *pApp = (CGndApp *)AfxGetApp();

  for (i=0; i<MAXDIRS; i++)
  {
        PFI[i].Dir = pApp->GetProfileString(strSectionDir, def[i]);
        PFI[i].Ext = pApp->GetProfileString(strSectionExt, def[i]);
        PFI[i].Des = pApp->GetProfileString(strSectionDscrpt, def[i]);
  }

  m_bTabDlgUp=FALSE;
}

CGndView::~CGndView()
{
  int i, imax;
  CGndDoc *pDoc = (CGndDoc *)GetDocument();

  // destroy tabbed dialog
  if (m_bTabDlgUp)
```

```
        {
                m_pTabDlg->DestroyWindow();
                delete m_pTabDlg;
                m_bTabDlgUp = FALSE;
        }


        // destroy all loaded macros stored in pDoc->m. ~SMacro takes care of all necessary stuff.
        for (i=0, imax=pDoc->m.GetUpperBound(); i<=imax; i++)
                delete (SMacro *)pDoc->m.GetAt(i);


        // destroy all built-in commands stored in pDoc->c.
        for (i=0, imax=pDoc->c.GetUpperBound(); i<=imax; i++)
                delete (SMacro *)pDoc->c.GetAt(i);
}


/////////////////////////////////////////////////////////////////////////////
// CGndView drawing

void CGndView::OnDraw(CDC* pDC)
{
        CGndDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);


        // TODO: add draw code for native data here
}


/////////////////////////////////////////////////////////////////////////////
// CGndView printing

BOOL CGndView::OnPreparePrinting(CPrintInfo* pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}


void CGndView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add extra initialization before printing
}


void CGndView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add cleanup after printing
}


/////////////////////////////////////////////////////////////////////////////
// CGndView diagnostics
```

```
#ifdef _DEBUG
void CGndView::AssertValid() const
{
        CView::AssertValid();
}


void CGndView::Dump(CDumpContext& dc) const
{
        CView::Dump(dc);
}


CGndDoc* CGndView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CGndDoc)));
        return (CGndDoc*)m_pDocument;
}
#endif //_DEBUG


/////////////////////////////////////////////////////////////////////////////
// CGndView message handlers

void CGndView::OnUserAccess()
{
  CPasswordDlg dlg;
  if (dlg.DoModal()==IDOK)
  {
    if (!m_bTabDlgUp)
    {
      // initialize controls in child dialogs
      // MyDataExchange(TRUE);

      // allocate your tab dialog object
      // pass in parent window
      m_pTabDlg = (CMainTabDlg*) new CMainTabDlg(this);

      // add child tabs to tab dialog internal list
      m_pTabDlg->AddChildDialog(CCHScriptsDlg::IDD,     (CTabDlgChild *)&m_ChDlg0);
      m_pTabDlg->AddChildDialog(CCHTelemetryDlg::IDD,   (CTabDlgChild *)&m_ChDlg1);
      m_pTabDlg->AddChildDialog(CCHMailDlg::IDD,        (CTabDlgChild *)&m_ChDlg2);
      m_pTabDlg->AddChildDialog(CCHMemoryDlg::IDD,      (CTabDlgChild *)&m_ChDlg3);
      m_pTabDlg->AddChildDialog(CCHControlDlg::IDD,     (CTabDlgChild *)&m_ChDlg4);
      m_pTabDlg->AddChildDialog(CCHOSControlDlg::IDD,   (CTabDlgChild *)&m_ChDlg5);
      m_pTabDlg->AddChildDialog(CCHFileSystemDlg::IDD,  (CTabDlgChild *)&m_ChDlg6);
      m_pTabDlg->AddChildDialog(CCHTaskControlDlg::IDD, (CTabDlgChild *)&m_ChDlg7);

      m_ChDlg0.pPFI = &PFI;
```

```
        // fire off tab dialog
        // if you didn't pass the parent window to the constructor
        // you should pass in the parent window as the second parameter here
        m_pTabDlg->DoModeless(CMainTabDlg::IDD, this);

        m_bTabDlgUp = TRUE;
    }
  }
}


void CGndView::OnEndUserAccess()
{
  if (m_bTabDlgUp)
  {
    m_pTabDlg->DestroyWindow();
    delete m_pTabDlg;
    Invalidate();
    m_bTabDlgUp = FALSE;
  }
}


void CGndView::OnPreferences()
{
  int i;

  CPrefDlg dlg;       // constructor call
  dlg.pOldPFI = &PFI; // initialize dlg data

  CGndApp *pApp = (CGndApp *)AfxGetApp(); // update INI settings from Gnd.INI
  for (i=0; i<MAXDIRS; i++)
  {
        PFI[i].Dir = pApp->GetProfileString(strSectionDir, def[i]);
        PFI[i].Ext = pApp->GetProfileString(strSectionExt, def[i]);
  }

  if ((dlg.DoModal() == IDOK) && (dlg.m_bHasChanged))
  {
    for (i=0; i<MAXDIRS; i++)  // user pressed OK: Directory setting validation
    {
        pApp->WriteProfileString(strSectionDir, def[i], dlg.NewPFI[i].Dir);
    }
  }
}



void CGndView::OnInitialUpdate()
{
```

110

```
    int i;
    char buf[5]; // enough for 10,000 macros
    CString strMPath;
    strSectionMacro = "Macro";

    CGndDoc *pDoc = (CGndDoc *)GetDocument();
    CGndApp *pApp = (CGndApp *)AfxGetApp();

    // load all macros referenced in *.INI to pDoc->m. The SMacro constructor actually loads 'em.
    for (i=0; ; i++)
    {
      strMPath = pApp->GetProfileString(strSectionMacro, itoa(i, buf, 10),";");
      if (!strcmp(LPCTSTR(strMPath),";")) break;
          pDoc->m.Add(new SMacro(LPCTSTR(strMPath)));
    }
    // get all built-in commands to pDoc->c
    for (i=0; i<nAmountCmd; i++)
      pDoc->c.Add(new SMacro(i));

    CView::OnInitialUpdate();
}
```

MainFrm.h

```
// mainfrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        //}}AFX_VIRTUAL

// Implementation
```

111

```
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

//

MainFrm.cpp

```
// mainfrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Gnd.h"

#include "gnddoc.h"
#include "mainfrm.h"
#include <stdio.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
```

```
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


///////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
        // same order as in the bitmap 'toolbar.bmp'
        ID_FILE_NEW,
        ID_FILE_OPEN,
        ID_FILE_SAVE,
                ID_SEPARATOR,
        ID_EDIT_CUT,
        ID_EDIT_COPY,
        ID_EDIT_PASTE,
                ID_SEPARATOR,
        ID_FILE_PRINT,
        ID_APP_ABOUT,
};

static UINT BASED_CODE indicators[] =
{
        ID_SEPARATOR,           // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};


///////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
```

```
                  !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
                  !m_wndToolBar.SetButtons(buttons,
                    sizeof(buttons)/sizeof(UINT)))
          {
                  TRACE0("Failed to create toolbar\n");
                  return -1;      // fail to create
          }

          if (!m_wndStatusBar.Create(this) ||
                  !m_wndStatusBar.SetIndicators(indicators,
                    sizeof(indicators)/sizeof(UINT)))
          {
                  TRACE0("Failed to create status bar\n");
                  return -1;      // fail to create
          }

          // TODO: Delete these three lines if you don't want the toolbar to
          //  be dockable
          m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
          EnableDocking(CBRS_ALIGN_ANY);
          DockControlBar(&m_wndToolBar);

          // TODO: Remove this if you don't want tool tips
          m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
                  CBRS_TOOLTIPS | CBRS_FLYBY);
/*
  CMenu *pMenu = GetMenu()->GetSubMenu(1);
  pMenu->EnableMenuItem(ID_ACCESS_LOGOFF, MF_GRAYED | MF_BYCOMMAND);
*/
          return 0;
}


/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
          CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
          CFrameWnd::Dump(dc);
}


#endif //_DEBUG
```

114

```
/////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

B. DIALOG SOURCECODE

Every groundstation child dialog has its own include (*.h) and implementation (*.cpp) file. Except the Script dialog (*ChScript*) all other dialogs do not have any special code besides the MSVC provided framework yet. That is why they are not listed here (*ChTlmtry*, *ChMail*, *ChMemory*, *ChOSCtrl*, *ChCtrl*, *ChFileSy*, *ChTaskCt*, *MainTab*).

ChScript.h	Include file for Scripts tabbed dialog
ChScript.cpp	Implementation file for Scripts tabbed dialog
MyTabDlg.h	Include file for all child dialog include (*.h) files

ChScript.h

```
// chscript.h : header file
//


/////////////////////////////////////////////////////////////////////////////
// CCHScriptsDlg dialog

class CCHScriptsDlg : public CTabDlgChild
{
// Construction
public:
        CCHScriptsDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
        //{{AFX_DATA(CCHScriptsDlg)
        enum { IDD = IDD_CH_SCRIPTS };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA
  struct PANSATFileInfo (*pPFI)[MAXDIRS];
  CString m_strFilter, m_strGndDir;
  WORD m_hlEditMode;  // hl=HotLinked
  WORD m_hlScriptType;
  long m_nCurCode;
  SMacro m_ActualMacro;
  OPENFILENAME m_ofn;

// Overrides
```

```cpp
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CCHScriptsDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
  void NewMacro();
protected:

        // Generated message map functions
        //{{AFX_MSG(CCHScriptsDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnEraseEditline();
        afx_msg void OnCutToEditline();
        afx_msg void OnInsertEditline();
        afx_msg void OnInsert();
        afx_msg void OnEdit();
        afx_msg void OnLoad();
        afx_msg void OnSave();
        afx_msg void OnSaveAs();
        afx_msg void OnNew();
        afx_msg void OnDelete();
        afx_msg void OnLClickListCmd();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

ChScript.cpp

```cpp
// chscript.cpp : implementation file
//

#include "stdafx.h"
#include "Gnd.h"
#include "chscript.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

#define TYPE_SCRIPT 1
#define TYPE_MACRO  2


/////////////////////////////////////////////////////////////////////////
// CCHScriptsDlg dialog
```

```
CCHScriptsDlg::CCHScriptsDlg(CWnd* pParent /*=NULL*/)
        : CTabDlgChild(CCHScriptsDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CCHScriptsDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
  m_hlEditMode   = 0;
  m_hlScriptType = 0;
}


void CCHScriptsDlg::DoDataExchange(CDataExchange* pDX)
{
        CTabDlgChild::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CCHScriptsDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CCHScriptsDlg, CTabDlgChild)
        //{{AFX_MSG_MAP(CCHScriptsDlg)
    ON_COMMAND(IDC_SCRIPT_ERASELINE, OnEraseEditline)
    ON_COMMAND(IDC_SCRIPT_CUTTOEL, OnCutToEditline)
    ON_COMMAND(IDC_SCRIPT_INSERTEL, OnInsertEditline)
    ON_COMMAND(IDC_SCRIPT_INSERT, OnInsert)
    ON_COMMAND(IDC_SCRIPT_EDIT, OnEdit)
    ON_COMMAND(IDC_SCRIPT_LOAD, OnLoad)
    ON_COMMAND(IDC_SCRIPT_SAVE, OnSave)
    ON_COMMAND(IDC_SCRIPT_SAVEAS, OnSaveAs)
    ON_COMMAND(IDC_SCRIPT_NEW, OnNew)
    ON_COMMAND(IDC_SCRIPT_DELETE, OnDelete)
    ON_CONTROL(WM_LBUTTONDOWN, IDC_SCRIPT_LISTCMD, OnLClickListCmd)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////
// CCHScriptsDlg message handlers

BOOL CCHScriptsDlg::OnInitDialog()
{
    CTabDlgChild::OnInitDialog();

    int i;
```

118

```
CHBRadio  *pEditMode  = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_NORMALEDIT);
CHBRadio  *pScriptType = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_SCRIPT);
CHList    *pCmdList    = (CHList    *)GetDlgItem(IDC_SCRIPT_LISTCMD);
CHList    *pScriptList = (CHList    *)GetDlgItem(IDC_SCRIPT_LISTSCRIPT);
CHEdit_SS *pEditCmd    = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_EDITCMD);
CHEdit_SS *pEditP      = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_P);

pEditMode->SetDataLink(TRUE, &m_hlEditMode);
pEditMode->SetState(TRUE, FALSE);
pScriptType->SetDataLink(TRUE, &m_hlScriptType);
pScriptType->SetState(TRUE, FALSE);
for (i=0; i<nAmountCmd; i++)
  pCmdList->AddItem(DocCmd[i].command);
pCmdList->SelectData(DocCmd[0].command, HL_SELECT);
pScriptList->AddItem("(next)");
pScriptList->SelectData("(next)", HL_SELECT);
pEditCmd->SetWindowText("");
pEditP->SetWindowText("");

m_strGndDir = "D:\\Ground";
m_strFilter = "All Files (*.*)";
m_strFilter += '\0';
m_strFilter += "*.*";
m_strFilter += '\0';

for (i=0; i<MAXDIRS; i++)
{
  m_strFilter += (*pPFI)[i].Des;
  m_strFilter += '\0';
  m_strFilter += (*pPFI)[i].Ext;
  m_strFilter += '\0';
}

m_strFilter += '\0';

m_ofn.lpstrFilter = LPCTSTR(m_strFilter);
m_ofn.lStructSize = sizeof(m_ofn);
m_ofn.hwndOwner = m_hWnd;
m_ofn.hInstance = NULL;//
m_ofn.lpstrCustomFilter = NULL;//
m_ofn.nMaxCustFilter = 0;//
m_ofn.nFileOffset = 0;//
m_ofn.nFileExtension = 0;//
m_ofn.lCustData = 0;//
m_ofn.lpfnHook = NULL;//
m_ofn.lpTemplateName = NULL;//
```

```
    return TRUE;   // return TRUE unless you set the focus to a control
                   // EXCEPTION: OCX Property Pages should return FALSE
}


void CCHScriptsDlg::OnEraseEditline()
{
  CHEdit_SS *pEditCmd  = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_EDITCMD);
  CHEdit_SS *pEditP    = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_P);

  pEditCmd->SetWindowText("");
  pEditP->SetWindowText("");
}


void CCHScriptsDlg::OnInsertEditline()
{
  CString str;
  int n, ix;
  CHList    *pScriptList = (CHList    *)GetDlgItem(IDC_SCRIPT_LISTSCRIPT);
  CHEdit_SS *pEditCmd    = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_EDITCMD);
  CHEdit_SS *pEditP      = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_P);
  CHBRadio  *pScriptType = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_SCRIPT);
  CHBRadio  *pMacroType  = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_MACRO);

  pEditCmd->GetWindowText(str);
  ix = pScriptList->InsertItem(pScriptList->GetCurSel(), (void *)LPCTSTR(str));
  pScriptList->SetCurSel(pScriptList->GetCurSel()-1);

  for (n=0; n<nAmountCmd, strcmp(DocCmd[n].command, str)!=0; n++);
  m_ActualMacro.cmd.InsertAt(ix, new SCmd(n)); //CPtrArray: InsertAt()
  m_ActualMacro.m_bHasChanged=TRUE;
  if (m_ActualMacro.IsScript()) pScriptType->SetState(TRUE, TRUE);
  else pMacroType->SetState(TRUE, TRUE);
}


void CCHScriptsDlg::OnCutToEditline()
{
  char str[40];
  int ix;
  CHList    *pScriptList = (CHList    *)GetDlgItem(IDC_SCRIPT_LISTSCRIPT);
  CHEdit_SS *pEditCmd    = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_EDITCMD);
  CHEdit_SS *pEditP      = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_P);
  CHBRadio  *pScriptType = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_SCRIPT);
  CHBRadio  *pMacroType  = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_MACRO);

  pScriptList->GetCurData(str,39);
  if (strcmp(str, "(next)"))
  {
```

```
      pScriptList->DeleteItem(ix = pScriptList->GetCurSel());
      pScriptList->SetCurSel(pScriptList->GetCurSel()+1);
      pEditCmd->SetWindowText(str);

      m_ActualMacro.cmd.RemoveAt(ix); //CPtrArray: RemoveAt()
      m_ActualMacro.m_bHasChanged=TRUE;
      if (m_ActualMacro.IsScript()) pScriptType->SetState(TRUE, TRUE);
      else pMacroType->SetState(TRUE, TRUE);
   }
}


void CCHScriptsDlg::OnInsert()
{
   int ix, n;
   char str[40];
   CHList    *pCmdList    = (CHList    *)GetDlgItem(IDC_SCRIPT_LISTCMD);
   CHList    *pScriptList = (CHList    *)GetDlgItem(IDC_SCRIPT_LISTSCRIPT);
   CHEdit_SS *pEditCmd    = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_EDITCMD);
   CHEdit_SS *pEditP      = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_P);
   CHBRadio  *pScriptType = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_SCRIPT);
   CHBRadio  *pMacroType  = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_MACRO);

   pCmdList->GetCurData(str,39);
   pEditCmd->SetWindowText(str);
   for (n=0; n<nAmountCmd, strcmp(DocCmd[n].command, str)!=0; n++);
   if (DocCmd[n].wParam_Type==TVOID && DocCmd[n].lParam_Type==TVOID)
   {
      ix = pScriptList->InsertItem(pScriptList->GetCurSel(), str);
      pScriptList->SetCurSel(pScriptList->GetCurSel()-1);

      m_ActualMacro.cmd.InsertAt(ix, new SCmd(n)); //CPtrArray: InsertAt()
      m_ActualMacro.m_bHasChanged=TRUE;
      if (m_ActualMacro.IsScript()) pScriptType->SetState(TRUE, TRUE);
      else pMacroType->SetState(TRUE, TRUE);
   }
}


void CCHScriptsDlg::OnEdit()
{
   char str[40];
   CHList    *pCmdList    = (CHList    *)GetDlgItem(IDC_SCRIPT_LISTCMD);
   CHEdit_SS *pEditCmd    = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_EDITCMD);
   CHEdit_SS *pEditP      = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_P);

   pCmdList->GetCurData(str,39);
   pEditCmd->SetWindowText(str);
}
```

```
void CCHScriptsDlg::OnLoad()
{
  char buf[256];
  char fname[256];
  buf[0] = '\0';

  m_ofn.nFilterIndex = m_hlScriptType==TYPE_SCRIPT ? IX_SCRIPT+2 : IX_MACRO+2;
  m_ofn.lpstrFile = buf;
  m_ofn.nMaxFile = sizeof(buf);
  m_ofn.lpstrFileTitle = fname;
  m_ofn.nMaxFileTitle = sizeof(fname);
  m_ofn.lpstrInitialDir = LPCTSTR(m_hlScriptType==TYPE_SCRIPT ?
                                  (*pPFI)[IX_SCRIPT].Dir :
                                  (*pPFI)[IX_MACRO].Dir);
  m_ofn.lpstrTitle = "Load Macro";
  m_ofn.Flags = OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST | OFN_HIDEREADONLY;
  m_ofn.lpstrDefExt = LPCTSTR(m_hlScriptType==TYPE_SCRIPT ?
                              (*pPFI)[IX_SCRIPT].Ext :
                              (*pPFI)[IX_MACRO].Ext);

  OnNew();
  if (GetOpenFileName(&m_ofn))
    if (m_ActualMacro.Load() != NOERR)
      MessageBox("Error while loading macro!", "Load Macro", MB_OK);
}

void CCHScriptsDlg::OnSave()
{
  int nSaveResult, ioresult=NOERR;
  CString strText, strHeader;
  BOOL bIsScript;
  CHBRadio *pScriptType = (CHBRadio *)GetDlgItem(IDC_SCRIPT_SCRIPT);
  CHBRadio *pMacroType  = (CHBRadio *)GetDlgItem(IDC_SCRIPT_MACRO);

  if (bIsScript = m_ActualMacro.IsScript()) pScriptType->SetState(TRUE, TRUE);
  else pMacroType->SetState(TRUE, TRUE);

  if (m_ActualMacro.m_bHasFileName)
  {
    if (ioresult = m_ActualMacro.Save()==ERR_FILE_EXISTS)
    {
        strText  = ErrAry[ioresult] + CString("\nChoose OK to overwrite it.");
        strHeader = "Save " + m_ActualMacro.FileName;
        nSaveResult = MessageBox(strText, strHeader, MB_ICONINFORMATION | MB_OKCANCEL);
        if (nSaveResult==IDOK) ioresult = m_ActualMacro.Overwrite();
    }
```

```
    }
    else OnSaveAs();
    if (ioresult!=NOERR)
    {
      MessageBox(ErrAry[ioresult],"File Error", MB_OK);
      return;
    }
    m_ActualMacro.m_bHasChanged = FALSE;
}


void CCHScriptsDlg::OnSaveAs()
{
    char buf[256];
    char fname[256];
    buf[0] = '\0';
    BOOL bIsScript;
    CHBRadio  *pScriptType = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_SCRIPT);
    CHBRadio  *pMacroType  = (CHBRadio  *)GetDlgItem(IDC_SCRIPT_MACRO);

    if (bIsScript = m_ActualMacro.IsScript()) pScriptType->SetState(TRUE, TRUE);
    else pMacroType->SetState(TRUE, TRUE);

    m_ofn.nFilterIndex = bIsScript ? IX_SCRIPT+2 : IX_MACRO+2;
    m_ofn.lpstrFile = buf;
    m_ofn.nMaxFile = sizeof(buf);
    m_ofn.lpstrFileTitle = fname;
    m_ofn.nMaxFileTitle = sizeof(fname);
    m_ofn.lpstrInitialDir = LPCTSTR(bIsScript ?
                                    (*pPFI)[IX_SCRIPT].Dir :
                                    (*pPFI)[IX_MACRO].Dir);
    m_ofn.lpstrTitle = "Save Macro As...";
    m_ofn.Flags = OFN_OVERWRITEPROMPT | OFN_HIDEREADONLY;
    m_ofn.lpstrDefExt = LPCTSTR(bIsScript ?
                                (*pPFI)[IX_SCRIPT].Ext :
                                (*pPFI)[IX_MACRO].Ext);

    if (GetOpenFileName(&m_ofn))
    {
      m_ActualMacro.FileName = m_ofn.lpstrFile;
          m_ActualMacro.m_bHasFileName = TRUE;
          if (m_ActualMacro.Save() == NOERR) m_ActualMacro.m_bHasChanged = FALSE;
    }
}


void CCHScriptsDlg::NewMacro()
{
    int i, imax;
```

```
        CHList    *pScriptList = (CHList    *)GetDlgItem(IDC_SCRIPT_LISTSCRIPT);
        CHEdit_SS *pEditCmd    = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_EDITCMD);
        CHEdit_SS *pEditP      = (CHEdit_SS *)GetDlgItem(IDC_SCRIPT_P);


        for (i=0, imax=m_ActualMacro.cmd.GetUpperBound(); i<=imax; i++)
          delete ( (SCmd *)(m_ActualMacro.cmd.GetAt(i)) );
        m_ActualMacro.cmd.RemoveAll();
        m_ActualMacro.m_bHasChanged = FALSE;
        imax=m_ActualMacro.cmd.GetUpperBound();
        for (i=0, imax=pScriptList->GetCount()-2; i<=imax; i++)
          pScriptList->DeleteItem(0);
        pScriptList->SetCurSel(0);
        pEditCmd->SetWindowText("");
        pEditP->SetWindowText("");
}


void CCHScriptsDlg::OnNew()
{
        int nResult, ioresult=NOERR;
        CString strText, strHeader;

        if (m_ActualMacro.m_bHasChanged)
        {
          nResult = MessageBox("The current macro has been changed.\nDo you want to save it first?",
                               "Load or New Macro",MB_ICONQUESTION | MB_YESNOCANCEL);
          if (nResult== IDYES)
          {
            OnSave();
            NewMacro();
          }
          else if (nResult==IDNO)
            NewMacro();
        }
        else NewMacro();
}


void CCHScriptsDlg::OnDelete()
{
        char buf[256];
        char fname[256];
        buf[0] = '\0';
        CString strText;

        m_ofn.nFilterIndex = m_hlScriptType==TYPE_SCRIPT ? IX_SCRIPT+2 : IX_MACRO+2;
        m_ofn.lpstrFile = buf;
        m_ofn.nMaxFile = sizeof(buf);
        m_ofn.lpstrFileTitle = fname;
```

124

```
m_ofn.nMaxFileTitle = sizeof(fname);
m_ofn.lpstrInitialDir = LPCTSTR(m_hlScriptType==TYPE_SCRIPT ?
                                (*pPFI)[IX_SCRIPT].Dir :
                                (*pPFI)[IX_MACRO].Dir);
m_ofn.lpstrTitle = "Delete Macro";
m_ofn.Flags = OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST | OFN_HIDEREADONLY;
m_ofn.lpstrDefExt = LPCTSTR(m_hlScriptType==TYPE_SCRIPT ?
                                (*pPFI)[IX_SCRIPT].Ext :
                                (*pPFI)[IX_MACRO].Ext);


if (GetOpenFileName(&m_ofn))
{
        if (!DeleteFile(m_ofn.lpstrFile))
        {
          strText = CString("Couldn't delete ") + m_ofn.lpstrFile;
          MessageBox(strText, "Delete File", MB_ICONINFORMATION | MB_OK);
        }
  }
}


void CCHScriptsDlg::OnLClickListCmd()
{
}
```

MyTabDlg.h

```
// mytabdlg.h : include file for all tabbed dialog .h files      (see Gndview.h)
//
//                     class:          Member variable:
#include "maintab.h"   //  CMainTabDlg       m_TabDlg;


#include "chscript.h" //  CCHScriptsDlg     m_ChDlg0;
#include "chtlmtry.h" //  CCHTelemetryDlg   m_ChDlg1;
#include "chmail.h"   //  CCHMailDlg        m_ChDlg2;
#include "chmemory.h" //  CCHMemoryDlg      m_ChDlg3;
#include "chctrl.h"   //  CCHControlDlg     m_ChDlg4;
#include "chosctrl.h" //  CCHOSControlDlg   m_ChDlg5;
#include "chfilesy.h" //  CCHFileSystemDlg  m_ChDlg6;
#include "chtaskct.h" //  CCHTaskControlDlg m_ChDlg7;
```

C.   MISCELLANEOUS

Gnd.RC   Resource file for Dialog Editor

Resource.h	Resource variables definition include file
Password.h	Include file for Password dialog
Password.cpp	Implementation file for Password dialog
PrefDlg.h	Include file for Preferences dialog
PrefDlg.cpp	Implementation file for Preferences dialog

Gnd.RC

```
//Microsoft Visual C++ generated resource script.
//
#include "mfcwidg.h"
#include "resource.h"


/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"


/////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS


#ifdef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""res\\Gnd.rc2""  // non-Microsoft Visual C++ edited resources\r\n"
```

```
                "\r\n"
                "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
                "#define _AFX_NO_OLE_RESOURCES\r\n"
                "#define _AFX_NO_TRACKER_RESOURCES\r\n"
                "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
                "#include ""afxres.rc""  \011// Standard components\r\n"
                "#include ""afxprint.rc""\011// printing/print preview resources\r\n"
                "\0"
        END


        /////////////////////////////////////////////////////////////////////////
        #endif    // APSTUDIO_INVOKED



        /////////////////////////////////////////////////////////////////////////
        //
        // Icon
        //


        IDR_MAINFRAME           ICON    DISCARDABLE     "res\\Gnd.ico"


        /////////////////////////////////////////////////////////////////////////
        //
        // Bitmap
        //


        IDR_MAINFRAME           BITMAP  MOVEABLE PURE   "res\\toolbar.bmp"


        /////////////////////////////////////////////////////////////////////////
        //
        // Menu
        //


        IDR_MAINFRAME MENU PRELOAD DISCARDABLE
        BEGIN
            POPUP "&File"
            BEGIN
                MENUITEM "&New\tCtrl+N",            ID_FILE_NEW
                MENUITEM "&Open...\tCtrl+O",        ID_FILE_OPEN
                MENUITEM "&Save\tCtrl+S",           ID_FILE_SAVE
                MENUITEM "Save &As...",             ID_FILE_SAVE_AS
                MENUITEM SEPARATOR
                MENUITEM "&Print...\tCtrl+P",       ID_FILE_PRINT
                MENUITEM "Print Pre&view",          ID_FILE_PRINT_PREVIEW
                MENUITEM "P&rint Setup...",         ID_FILE_PRINT_SETUP
                MENUITEM SEPARATOR
                MENUITEM "Recent File",             ID_FILE_MRU_FILE1, GRAYED
```

```
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                    ID_APP_EXIT
    END
    POPUP "Access"
    BEGIN
        MENUITEM "Logon",                    ID_ACCESS_LOGON
        MENUITEM "Logoff",                   ID_ACCESS_LOGOFF, GRAYED
    END
    MENUITEM "Preferences",              ID_PREFERENCES
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                 ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",              ID_VIEW_STATUS_BAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About Gnd...",            ID_APP_ABOUT
    END
    POPUP "Debug"
    BEGIN
        MENUITEM "Load Macro",               ID_DEBUG_LOADMACRO
    END
END


/////////////////////////////////////////////////////////////////////////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
    "C",            ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    "N",            ID_FILE_NEW,            VIRTKEY, CONTROL, NOINVERT
    "O",            ID_FILE_OPEN,           VIRTKEY, CONTROL, NOINVERT
    "P",            ID_FILE_PRINT,          VIRTKEY, CONTROL, NOINVERT
    "S",            ID_FILE_SAVE,           VIRTKEY, CONTROL, NOINVERT
    "V",            ID_EDIT_PASTE,          VIRTKEY, CONTROL, NOINVERT
    VK_BACK,        ID_EDIT_UNDO,           VIRTKEY, ALT, NOINVERT
    VK_DELETE,      ID_EDIT_CUT,            VIRTKEY, SHIFT, NOINVERT
    VK_F5,          IDC_SCRIPT_CUTTOEL,     VIRTKEY, NOINVERT
    VK_F6,          ID_NEXT_PANE,           VIRTKEY, NOINVERT
    VK_F6,          ID_PREV_PANE,           VIRTKEY, SHIFT, NOINVERT
    VK_INSERT,      ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    VK_INSERT,      ID_EDIT_PASTE,          VIRTKEY, SHIFT, NOINVERT
    "X",            ID_EDIT_CUT,            VIRTKEY, CONTROL, NOINVERT
    "Z",            ID_EDIT_UNDO,           VIRTKEY, CONTROL, NOINVERT
```

```
END


///////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  34, 22, 217, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About Gnd"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,20,20
    LTEXT           "Gnd Version 1.0",IDC_STATIC,40,10,119,8
    LTEXT           "Copyright Jens Bartschat\251 1995",IDC_STATIC,40,25,119, 8
    DEFPUSHBUTTON   "OK",IDOK,176,6,32,14,WS_GROUP
END


IDD_CH_SCRIPTS DIALOG 18, 18, 397, 249
STYLE WS_CHILD
FONT 8, "MS Sans Serif"
{
 CONTROL "%ssHList", IDC_SCRIPT_LISTSCRIPT, "HList", HLS_BORDER3D | HLS_NONINTHEIGHT | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 6,62,161,161
 CONTROL "333;Normal Editing;HR1:HR2:HR3:HR2", IDC_SCRIPT_NORMALEDIT, "HButt", HBS_RADIOBUTTON | HBS_TRANSPARENT |
HBS_LJUST | HBS_DOWNPICS | HBS_AUTOADVANCE | HBS_NOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 12,10, 56,12
 CONTROL "333;Express Editing;HR1:HR2:HR3:HR2", IDC_SCRIPT_EXPRESSEDIT, "HButt", HBS_RADIOBUTTON | HBS_TRANSPARENT |
HBS_LJUST | HBS_DOWNPICS | HBS_AUTOADVANCE | HBS_NOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 12,24, 58,12
 CONTROL "HStat", IDC_STATIC, "HStat", HSS_FRAME | HSS_BUMP | HSS_TRANSPARENT | WS_CHILD | WS_VISIBLE, 6,7,68,32
 CONTROL "433;Script;HR1:HR2:HR3:HR2", IDC_SCRIPT_SCRIPT, "HButt", HBS_RADIOBUTTON | HBS_TRANSPARENT | HBS_LJUST |
HBS_DOWNPICS | HBS_AUTOADVANCE | HBS_NOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 85,10,38,12
 CONTROL "433;Macro;HR1:HR2:HR3:HR2", IDC_SCRIPT_MACRO, "HButt", HBS_RADIOBUTTON | HBS_TRANSPARENT | HBS_LJUST |
HBS_DOWNPICS | HBS_AUTOADVANCE | HBS_NOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 85,24,39,12
 CONTROL "HStat", IDC_STATIC, "HStat", HSS_FRAME | HSS_BUMP | HSS_LEFT | HSS_TRANSPARENT | WS_CHILD | WS_VISIBLE,
78,7,44,32
 CONTROL "441;Insert from Edit Line;", IDC_SCRIPT_INSERTEL, "HButt", HBS_LJUST | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
6,229,69,14
 CONTROL "441;Cut to Edit Line;", IDC_SCRIPT_CUTTOEL, "HButt", HBS_RJUST | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
98,229,69,14
 CONTROL "441;Insert;", IDC_SCRIPT_INSERT, "HButt", HBS_LJUST | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 191,229,37,14
 CONTROL "441;Edit;", IDC_SCRIPT_EDIT, "HButt", HBS_RJUST | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 250,229,37,14
 CONTROL "%ssHList", IDC_SCRIPT_LISTCMD, "HList", HLS_BORDER3D | HLS_NONINTHEIGHT | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 191,62,96,161
 CONTROL "%sspurge_stored_telemetry", IDC_SCRIPT_EDITCMD, "HEdit_SS", HES_AUTOHSCROLL | HES_BORDER3D | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 6,47,86,12
 CONTROL "441;Erase Edit Line;", IDC_SCRIPT_ERASELINE, "HButt", WS_CHILD | WS_VISIBLE | WS_TABSTOP, 293, 47, 52, 12
```

129

```
    CONTROL "441;Load...;", IDC_SCRIPT_LOAD, "HButt", HBS_LJUST | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 325, 80, 50, 14
    CONTROL "441;Save;", IDC_SCRIPT_SAVE, "HButt", HBS_LJUST | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 325, 109, 50, 14
    CONTROL "441;Save As...;", IDC_SCRIPT_SAVEAS, "HButt", HBS_LJUST | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 325, 138,
50, 14
    CONTROL "441;New;", IDC_SCRIPT_NEW, "HButt", HBS_LJUST | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 325, 167, 50, 14
    CONTROL "441;Delete;", IDC_SCRIPT_DELETE, "HButt", HBS_LJUST | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 325,209,50,14
    CONTROL "%ssHEdit_SS", IDC_SCRIPT_P, "HEdit_SS", HES_AUTOHSCROLL | HES_BORDER3D | HES_READONLY | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 94, 47, 193, 12
}


IDD_CH_CONTROL DIALOG DISCARDABLE  18, 18, 397, 249
STYLE WS_CHILD
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "RF System",IDC_STATIC,"HStat",0x251,94,7,110,167
    CONTROL         "Receiver",IDC_STATIC,"HStat",0x211,98,22,49,75
    CONTROL         "Transmitter",IDC_STATIC,"HStat",0x211,151,22,49,75
    CONTROL         "441;Mix\n#5;",7000,"HButt",WS_GROUP | 0xc86,102,41,21, 22
    CONTROL         "441;Mix\n#6;",7001,"HButt",0xc86,122,41,21,22
    CONTROL         "441;Mix\n#5;",7002,"HButt",WS_GROUP | 0xc86,155,41,21, 22
    CONTROL         "441;Mix\n#6;",7003,"HButt",0xc86,175,41,21,22
    CONTROL         "441;LNA\n#1;",7004,"HButt",WS_GROUP | 0xc86,102,71,21, 22
    CONTROL         "441;LNA\n#2;",7005,"HButt",0xc86,122,71,21,22
    CONTROL         "441;HPA\n#3;",7006,"HButt",WS_GROUP | 0xc86,155,71,21, 22
    CONTROL         "441;HPA\n#4;",7007,"HButt",0xc86,175,71,21,22
    CONTROL         "Power Level:",IDC_STATIC,"HStat",0x240,102,107,41,12
    CONTROL         "",7008,"HSpin",WS_TABSTOP | 0x280,145,106,29,12
    CONTROL         "dB",IDC_STATIC,"HStat",0x240,178,107,13,12
    CONTROL         "441;Spread\nSpectrum;",7009,"HButt",WS_GROUP | 0xc86, 102,144,47,22
    CONTROL         "441;Binary Phase\nKey Shifting;",7010,"HButt",0xc86,148, 144,48,22
    CONTROL         "A",IDC_STATIC,"HStat",0x211,212,22,50,64
    CONTROL         "B",IDC_STATIC,"HStat",0x211,266,22,50,64
    CONTROL         "441;Charge;",7013,"HButt",WS_GROUP | 0xca6,216,41,42,14
    CONTROL         "441;Discharge;",7014,"HButt",0xc86,216,54,42,14
    CONTROL         "441;Offline;",7015,"HButt",0xc86,216,67,42,14
    CONTROL         "441;Charge;",7016,"HButt",WS_GROUP | 0xca6,270,41,42,14
    CONTROL         "441;Discharge;",7017,"HButt",0xc86,270,54,42,14
    CONTROL         "441;Offline;",7018,"HButt",0xc86,270,67,42,14
    CONTROL         "441;Read;",7019,"HButt",WS_TABSTOP,99,197,45,14
    CONTROL         "441;Set;",7020,"HButt",WS_TABSTOP,154,197,45,14
    CONTROL         "PANSAT Clock",IDC_STATIC,"HStat",0x251,94,177,110,67
    CONTROL         "Batteries",IDC_STATIC,"HStat",0x251,208,7,112,83
    CONTROL         "Transmit Mode",IDC_STATIC,"HStat",0x211,98,125,102,45
    CONTROL         "Watchdog",IDC_STATIC,"HStat",0x251,208,94,112,80
    CONTROL         "441;Reset;",7021,"HButt",WS_TABSTOP,214,141,45,28
    CONTROL         "441;Stop;",7022,"HButt",WS_TABSTOP,269,141,45,28
    CONTROL         "333;DCS #1;HR1:HR2:HR3:HR2",7023,"HButt",WS_TABSTOP | 0x1c33,218,109,50,12
```

130

```
    CONTROL         "333;DCS #2;HR1:HR2:HR3:HR2",7024,"HButt",WS_TABSTOP | 0x1c33,218,123,50,12
    CONTROL         "%aqhh:mm:ss Ddd, Mmm dd,yy",7025,"HEdit_SS",0x201,99, 222,101,15
    CONTROL         "441;On;",7029,"HButt",WS_GROUP | 0xc86,43,25,21,22
    CONTROL         "441;Off;",7030,"HButt",0xc86,63,25,21,22
    CONTROL         "441;On;",7031,"HButt",WS_GROUP | 0xc86,43,55,21,22
    CONTROL         "441;Off;",7032,"HButt",0xc86,63,55,21,22
    CONTROL         "441;On;",7033,"HButt",WS_GROUP | 0xc86,43,85,21,22
    CONTROL         "441;Off;",7034,"HButt",0xc86,63,85,21,22
    CONTROL         "441;On;",7035,"HButt",WS_GROUP | 0xc86,43,115,21,22
    CONTROL         "441;Off;",7036,"HButt",0xca6,63,115,21,22
    CONTROL         "441;On;",7037,"HButt",WS_GROUP | 0xc86,43,145,21,22
    CONTROL         "441;Off;",7038,"HButt",0xc86,63,145,21,22
    CONTROL         "RF:",IDC_STATIC,"HStat",0x280,19,31,21,12
    CONTROL         "MUX A:",IDC_STATIC,"HStat",0x280,15,61,25,12
    CONTROL         "MUX B:",IDC_STATIC,"HStat",0x280,15,91,25,12
    CONTROL         "MStor A:",IDC_STATIC,"HStat",0x280,10,121,30,12
    CONTROL         "MStor B:",IDC_STATIC,"HStat",0x280,12,151,28,12
    CONTROL         "HStat",IDC_STATIC,"HStat",0x212,10,142,76,28
    CONTROL         "HStat",IDC_STATIC,"HStat",0x212,10,22,76,28
    CONTROL         "HStat",IDC_STATIC,"HStat",0x212,10,52,76,28
    CONTROL         "HStat",IDC_STATIC,"HStat",0x212,10,82,76,28
    CONTROL         "HStat",IDC_STATIC,"HStat",0x212,10,112,76,28
    CONTROL         "Power Switches",IDC_STATIC,"HStat",0x251,6,7,84,167
    CONTROL         "SCOS Parameters",IDC_STATIC,"HStat",0x251,6,177,84,67
    CONTROL         "441;Read;",7040,"HButt",WS_TABSTOP,23,197,50,14
    CONTROL         "441;Update;",7041,"HButt",WS_TABSTOP,23,220,50,14
    CONTROL         "Warm Boot DCS",IDC_STATIC,"HStat",0x251,323,94,68,80
    CONTROL         "333;DCS #1;HR1:HR2:HR3:HR2",7042,"HButt",WS_TABSTOP | 0x1c33,340,109,36,12
    CONTROL         "333;DCS #2;HR1:HR2:HR3:HR2",7043,"HButt",WS_TABSTOP | 0x1c33,340,123,35,12
    CONTROL         "441;ROM\nBoot;",7044,"HButt",WS_TABSTOP,335,141,45,28
    CONTROL         "Temperature MUX",7011,"HStat",0x251,323,7,68,83
    CONTROL         "Peripheral\nControl Bus",IDC_STATIC,"HStat",0x251,323, 177,68,67
    CONTROL         "441;Init;",7026,"HButt",WS_TABSTOP,335,211,45,28
END


IDD_MAIN DIALOG DISCARDABLE  42, 27, 441, 311
STYLE DS_MODALFRAME | WS_MINIMIZEBOX | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Main Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "%kThh:mm:ss Ddd, Mmm dd,yy",IDC_SYSTEM_TIME,"HEdit_SS", 0x201,102,289,101,15
    CONTROL         "%tihh:mm:ss",IDC_STOPWATCH,"HEdit_SS",WS_TABSTOP | 0x281,214,289,48,15
    PUSHBUTTON      "",IDC_STATIC,0,0,400,265,NOT WS_TABSTOP
    CONTROL         "[DkRed];[64,0,0];HStat",IDC_SEND,"HStat",0x22,39,272,26, 14
    CONTROL         "[DkBlue];[0,0,64];HStat",IDC_RECEIVE,"HStat",0x22,39, 291,26,14
    CONTROL         "Send",106,"HStat",0x280,4,274,30,12
    CONTROL         "Receive",107,"HStat",0x280,4,293,30,12
```

```
        CONTROL         "System Time && Date:",IDC_STATIC,"HStat",0x240,102,275, 69,12
        CONTROL         "444;Start;",IDC_START_STOPWATCH,"HButt",WS_TABSTOP,214, 272,21,14
        CONTROL         "441;Pause;",IDC_PAUSE_STOPWATCH,"HButt",WS_GROUP |  0xc86,239,272,23,14
        CONTROL         "441;0;",IDC_MACRO_0,"HButt",WS_TABSTOP,400,0,41,15
        CONTROL         "441;1;",IDC_MACRO_1,"HButt",WS_TABSTOP,400,15,41,15
        CONTROL         "441;2;",IDC_MACRO_2,"HButt",WS_TABSTOP,400,30,41,15
        CONTROL         "441;3;",IDC_MACRO_3,"HButt",WS_TABSTOP,400,45,41,15
        CONTROL         "441;4;",IDC_MACRO_4,"HButt",WS_TABSTOP,400,60,41,15
        CONTROL         "441;5;",IDC_MACRO_5,"HButt",WS_TABSTOP,400,75,41,15
        CONTROL         "441;6;",IDC_MACRO_6,"HButt",WS_TABSTOP,400,90,41,15
        CONTROL         "441;7;",IDC_MACRO_7,"HButt",WS_TABSTOP,400,105,41,15
        CONTROL         "441;8;",IDC_MACRO_8,"HButt",WS_TABSTOP,400,120,41,15
        CONTROL         "441;9;",IDC_MACRO_9,"HButt",WS_TABSTOP,400,135,41,15
        CONTROL         "441;10;",IDC_MACRO_10,"HButt",WS_TABSTOP,400,150,41,15
        CONTROL         "441;11;",IDC_MACRO_11,"HButt",WS_TABSTOP,400,165,41,15
        CONTROL         "441;12;",IDC_MACRO_12,"HButt",WS_TABSTOP,400,180,41,15
        CONTROL         "441;13;",IDC_MACRO_13,"HButt",WS_TABSTOP,400,195,41,15
        CONTROL         "441;14;",IDC_MACRO_14,"HButt",WS_TABSTOP,400,210,41,15
        CONTROL         "441;15;",IDC_MACRO_15,"HButt",WS_TABSTOP,400,225,41,15
        CONTROL         "441;...;",IDC_MACRO_NEXT,"HButt",WS_TABSTOP,400,240,41, 25
        CONTROL         "%84%ss%ss",IDC_COMBO_USERLOG,"HComb",WS_TABSTOP | 0x111, 285,292,150,12
        CONTROL         "Log:",IDC_STATIC,"HStat",0x240,285,275,16,12
        CONTROL         "441;PCL;",IDC_LOG_PCL,"HButt",WS_GROUP | 0xc86,323,273, 23,14
        CONTROL         "441;User;",IDC_LOG_USER,"HButt",0xc86,345,273,23,14
        CONTROL         "441;MCL;",IDC_LOG_MACRO,"HButt",0xc86,367,273,23,14
END


IDD_CH_TELEMETRY DIALOG DISCARDABLE  18, 18, 397, 249
STYLE WS_CHILD | WS_BORDER
FONT 8, "MS Sans Serif"
BEGIN
END


IDD_CH_OSCONTROL DIALOG DISCARDABLE  18, 18, 397, 249
STYLE WS_CHILD
FONT 8, "MS Sans Serif"
BEGIN
        CONTROL         "User control, hahahaa!",IDC_STATIC,"HStat",0x262,7,6, 129,47
        CONTROL         "333;Drop && Lockout;HR1:HR2:HR3:HR2",8000,"HButt", WS_TABSTOP | 0x1c33,66,10,66,12
        CONTROL         "333;Lockout;HR1:HR2:HR3:HR2",8001,"HButt",WS_TABSTOP | 0x1c33,66,24,66,12
        CONTROL         "333;Unlock;HR1:HR2:HR3:HR2",8002,"HButt",WS_TABSTOP | 0x1c33,66,38,66,12
        CONTROL         "Foreign Users:",IDC_STATIC,"HStat",0x240,13,11,48,12
        CONTROL         "EVENTLOG",8003,"HGrid",WS_BORDER | WS_VSCROLL | WS_TABSTOP | 0x2b81,7,78,183,136
        CONTROL         "Event Log",IDC_STATIC,"HStat",0x240,7,63,37,12
        CONTROL         "441;Read;",8004,"HButt",WS_TABSTOP,31,224,50,14
        CONTROL         "441;Purge All;",8005,"HButt",WS_TABSTOP,112,224,50,14
        CONTROL         "%sshh:mm:ss ap Ddd, Mmm dd,yy",8006,"HEdit_SS", WS_TABSTOP | 0x280,94,62,96,12
```

```
        CONTROL         "Start:",8007,"HStat",0x240,75,63,20,12
        CONTROL         "441;Add...;",8008,"HButt",WS_TABSTOP,215,224,36,14
        CONTROL         "441;Delete;",8009,"HButt",WS_TABSTOP,258,224,36,14
        CONTROL         "441;List;",8010,"HButt",WS_TABSTOP,301,224,36,14
        CONTROL         "441;Purge All;",8011,"HButt",WS_TABSTOP | 0x20,344,224, 36,14
        CONTROL         "TIMETAG",8020,"HGrid",WS_BORDER | WS_VSCROLL | WS_TABSTOP | 0x2b81,206,78,183,136
        CONTROL         "Time-Tagged Commands",IDC_STATIC,"HStat",0x240,206,63, 84,12
END


IDD_CH_MAIL DIALOG DISCARDABLE  18, 18, 397, 249
STYLE WS_CHILD
FONT 8, "MS Sans Serif"
BEGIN
        CONTROL         "PANSAT Mail Directory:",IDC_STATIC,"HStat",0x240,6,81, 77,12
        CONTROL         "%ssHList",IDC_MAIL_LISTMAIL,"HList",WS_TABSTOP | 0x10,6, 96,91,147
        CONTROL         "%ssHList",IDC_MAIL_LISTMAILFILE,"HList",WS_TABSTOP | 0x110,202,23,189,220
        CONTROL         "%ssHEdit_SS",IDC_MAIL_MAILFILENAME,"HEdit_SS", WS_TABSTOP | 0x2280,202,6,91,12
        CONTROL         "441;Get Directory;",IDC_MAIL_GETDIR,"HButt",WS_TABSTOP | 0x20,115,119,55,14
        CONTROL         "441;Get Mail;",IDC_MAIL_READMSG,"HButt",WS_TABSTOP | 0x20,115,96,55,14
        CONTROL         "441;Add Mail;",IDC_MAIL_ADDMSG,"HButt",WS_TABSTOP | 0x20,115,229,55,14
        CONTROL         "441;Delete Mail;",IDC_MAIL_DELMSG,"HButt",WS_TABSTOP | 0x20,115,163,55,14
        CONTROL         "441;Purge All Mail;",IDC_MAIL_PURGEMSG,"HButt", WS_TABSTOP | 0x20,115,186,55,14
        CONTROL         "HEdit_SS",IDC_MAIL_FROM,"HEdit_SS",WS_TABSTOP | 0x280, 36,6,141,12
        CONTROL         "HEdit_SS",IDC_MAIL_TO,"HEdit_SS",WS_TABSTOP | 0x280,36, 22,141,12
        CONTROL         "%tImm/dd/yy hh:mm ap",IDC_MAIL_TIME,"HEdit_SS", WS_TABSTOP | 0x280,36,38,62,12
        CONTROL         "HEdit_SS",IDC_MAIL_SUBJECT,"HEdit_SS",WS_TABSTOP | 0x280,36,54,141,12
        CONTROL         "From:",IDC_STATIC,"HStat",0x280,6,7,26,12
        CONTROL         "To:",IDC_STATIC,"HStat",0x280,6,23,26,12
        CONTROL         "Date:",IDC_STATIC,"HStat",0x280,6,39,26,12
        CONTROL         "Subject:",IDC_STATIC,"HStat",0x280,6,55,26,12
END


IDD_CH_MEMORY DIALOG DISCARDABLE  18, 18, 397, 249
STYLE WS_CHILD | WS_BORDER
FONT 8, "MS Sans Serif"
BEGIN
        CONTROL         "HStat",IDC_STATIC,"HStat",0x262,93,7,112,40
        CONTROL         "333;RAM;HR1:HR2:HR3:HR2",9000,"HButt",WS_GROUP | WS_TABSTOP | 0x1c33,135,13,27,12
        CONTROL         "333;ROM;HR1:HR2:HR3:HR2",9001,"HButt",WS_TABSTOP | 0x1c33,135,30,26,12
        CONTROL         "333;SRAM;HR1:HR2:HR3:HR2",9002,"HButt",WS_TABSTOP | 0x1c33,170,13,31,12
        CONTROL         "333;FLASH;HR1:HR2:HR3:HR2",9004,"HButt",WS_TABSTOP | 0x1c33,170,30,32,12
        CONTROL         "HStat",IDC_STATIC,"HStat",0x262,209,7,174,40
        CONTROL         "441;Mass\nA;",9010,"HButt",WS_GROUP | 0xc86,215,13,28, 28
        CONTROL         "441;Mass\nB;",9011,"HButt",0xc86,242,13,28,28
        CONTROL         "441;AMUX\nA;",9012,"HButt",0xc86,269,13,28,28
        CONTROL         "441;AMUX\nB;",9013,"HButt",0xc86,296,13,28,28
        CONTROL         "441;EPS;",9014,"HButt",0xc86,323,13,28,28
```

```
    CONTROL         "441;RF\nSystem;",9015,"HButt",0xc86,350,13,28,28
    CONTROL         "HStat",IDC_STATIC,"HStat",0x262,6,7,83,40
    CONTROL         "333;20-Bit;HR1:HR2:HR3:HR2",9020,"HButt",WS_GROUP | WS_TABSTOP | 0x1c33,50,13,35,12
    CONTROL         "333;Seg:Off;HR1:HR2:HR3:HR2",9021,"HButt",WS_TABSTOP | 0x1c33,50,30,37,12
    CONTROL         "Memory:",IDC_STATIC,"HStat",0x240,99,14,28,12
    CONTROL         "Address:",104,"HStat",0x240,12,14,30,12
    CONTROL         "MEMEDIT",9040,"HGrid",WS_BORDER | WS_VSCROLL | WS_TABSTOP | 0x490a,6,67,261,176
    CONTROL         "441;Edit;",9050,"HButt",WS_GROUP | 0xc86,309,68,28,28
    CONTROL         "441;View;",9051,"HButt",0xc86,336,68,28,28
    CONTROL         "Memory Block",IDC_STATIC,"HStat",0x240,6,55,50,12
    CONTROL         "441;Write\nModified\nBytes;",9062,"HButt",WS_TABSTOP, 277,210,50,33
    CONTROL         "%nu|0|",107,"HEdit_SS",WS_TABSTOP | 0x2280,285,125,50,12
    CONTROL         "Bytes modified:",IDC_STATIC,"HStat",0x240,285,113,50,12
    CONTROL         "Bytes written:",IDC_STATIC,"HStat",0x240,285,153,50,12
    CONTROL         "%nu|0|",105,"HEdit_SS",WS_TABSTOP | 0x2280,285,165,50,12
    CONTROL         "441;Cancel;",106,"HButt",WS_TABSTOP,339,124,50,14
    CONTROL         "441;Reset;",108,"HButt",WS_TABSTOP | 0x20,339,164,50,14
    CONTROL         "441;Re-Read\nVisible\nBlock;",9063,"HButt",WS_TABSTOP,338,210,50,33
END


IDD_CH_FILESYSTEM DIALOG DISCARDABLE  18, 18, 397, 249
STYLE WS_CHILD
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "PANSAT File Directory:",IDC_STATIC,"HStat",0x240,7,6,77,12
    CONTROL         "%ssHList",IDC_FILE_LISTFILE,"HList",WS_TABSTOP | 0xa10,7,20,91,222
    CONTROL         "441;Read Directory;",IDC_FILE_GETDIR,"HButt",WS_TABSTOP | 0x20,116,118,55,14
    CONTROL         "441;Read File;",IDC_FILE_READFILE,"HButt",WS_TABSTOP | 0x20,116,95,55,14
    CONTROL         "441;Write File;",IDC_FILE_ADDFILE,"HButt",WS_TABSTOP | 0x20,116,228,55,14
    CONTROL         "441;Delete File;",IDC_FILE_DELFILE,"HButt",WS_TABSTOP | 0x20,116,162,55,14
    CONTROL         "441;Purge All Files;",IDC_FILE_PURGEFILE,"HButt", WS_TABSTOP | 0x20,116,185,55,14
    CONTROL         "Selected File(s):",IDC_STATIC,"HStat",0x240,201,6,77,12
    CONTROL         "%ssHList",IDC_FILE_SELECTFILE,"HList",WS_TABSTOP | 0xa10,201,36,91,206
    CONTROL         "HComb",IDC_FILE_SELECTCOMBO,"HComb",WS_TABSTOP | 0x119, 201,20,91,12
END


IDD_CH_TASKCONTROL DIALOG DISCARDABLE  18, 18, 397, 249
STYLE WS_CHILD
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "TASKCT",IDC_TASK_GRIDTASK,"HGrid",WS_BORDER | WS_VSCROLL | WS_TABSTOP | 0x2988,6,77,150,166
    CONTROL         "Add means...",IDC_STATIC,"HStat",0x262,6,7,129,47
    CONTROL         "333;Add && Start Task && Get List;HR1:HR2:HR3:HR2",
                    IDC_TASK_RADIOAUTO,"HButt",WS_TABSTOP | 0x1c33,36,11,97,12
    CONTROL         "333;Add && Get List;HR1:HR2:HR3:HR2",IDC_TASK_RADIOLIST,"HButt",WS_TABSTOP | 0x1c33,36,25,97,12
    CONTROL         "333;Add;HR1:HR2:HR3:HR2",IDC_TASK_RADIOADD,"HButt", WS_TABSTOP | 0x1c33,36,39,97,12
    CONTROL         "Add:",IDC_STATIC,"HStat",0x240,12,12,16,12
```

134

```
    CONTROL         "441;Add;",IDC_TASK_ADD,"HButt",WS_TABSTOP,164,77,69,14
    CONTROL         "441;Delete Task;",IDC_TASK_DELETE,"HButt",WS_TABSTOP,164,229,50,14
    CONTROL         "441;Get Tasklist;",IDC_TASK_GETLIST,"HButt",WS_TABSTOP,164,150,50,14
    CONTROL         "TASKLST",IDC_TASK_GRIDFILES,"HGrid",WS_BORDER | WS_VSCROLL | WS_TABSTOP | 0x2988,242,97,99,146
    CONTROL         "%ssHEdit_SS",IDC_TASK_EDITFILE,"HEdit_SS",WS_TABSTOP | 0x280,242,77,99,14
    CONTROL         "441;Load...;",IDC_TASK_LOAD,"HButt",WS_TABSTOP,349,97,41,14
    CONTROL         "PANSAT Task List",IDC_STATIC,"HStat",0x240,8,63,63,12
    CONTROL         "Available Task(s)",IDC_STATIC,"HStat",0x240,242,63,62,12
END


IDD_PREFERENCES DIALOG DISCARDABLE  0, 0, 186, 173
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Preferences"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,15,148,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,115,148,50,14
    EDITTEXT        IDC_EDIT1,64,15,112,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT2,64,31,112,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT3,64,47,112,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT4,64,63,112,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT5,64,79,112,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT6,64,95,112,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT7,64,111,112,13,ES_AUTOHSCROLL
    RTEXT           "Scripts:",IDC_STATIC,8,18,51,13
    RTEXT           "Macros:",IDC_STATIC,8,34,51,13
    RTEXT           "Telemetry Data:",IDC_STATIC,8,50,51,13
    RTEXT           "User Log:",IDC_STATIC,8,66,51,13
    RTEXT           "Task List:",IDC_STATIC,8,82,51,13
    RTEXT           "IN Data:",IDC_STATIC,8,98,51,13
    RTEXT           "OUT Data:",IDC_STATIC,8,114,51,13
    GROUPBOX        "Directory Settings",IDC_STATIC,5,1,176,137
END


IDD_USERLOGIN DIALOG DISCARDABLE  18, 18, 142, 92
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "PANSAT Groundstation User Login"
BEGIN
    CONTROL         "%ssjbartschat",IDC_LOGIN_LOGIN,"HEdit_SS",WS_TABSTOP | 0x280,56,28,77,12
    CONTROL         "%ss",IDC_LOGIN_PASSWORD,"HEdit_SS",WS_TABSTOP | 0x280,56,46,77,12
    CONTROL         "441;Ok;",IDOK,"HButt",WS_TABSTOP | 0x1,13,70,50,14
    CONTROL         "441;Cancel;",IDCANCEL,"HButt",WS_TABSTOP | 0x20,78,70,50,14
    CONTROL         "Login:",IDC_STATIC,"HStat",0x240,14,29,40,12
    CONTROL         "Password:",IDC_STATIC,"HStat",0x240,14,48,36,12
    CONTROL         "Please enter your login and password",IDC_STATIC,"HStat",0x240,10,10,125,12
END
```

```
/////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "CompanyName", "NPS SSAG (German branch)\0"
            VALUE "FileDescription", "PANSAT Groundstation\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "GND\0"
            VALUE "LegalCopyright", "Copyright \251 1995 Jens Bartschat\0"
            VALUE "OriginalFilename", "GND.EXE\0"
            VALUE "ProductName", "PANSAT Groundstation\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END


/////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
```

```
    IDR_MAINFRAME          "PANSAT Groundstation\n Groundstation\n Groundstation Document\n\n\n
Groundstation.Document\n Groundstation Document"
END


STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE      "PANSAT Groundstation"
    AFX_IDS_IDLEMESSAGE    "Ready"
END


STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT       "EXT"
    ID_INDICATOR_CAPS      "CAP"
    ID_INDICATOR_NUM       "NUM"
    ID_INDICATOR_SCRL      "SCRL"
    ID_INDICATOR_OVR       "OVR"
    ID_INDICATOR_REC       "REC"
END


STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW            "Create a new document\nNew"
    ID_FILE_OPEN           "Open an existing document\nOpen"
    ID_FILE_CLOSE          "Close the active document\nClose"
    ID_FILE_SAVE           "Save the active document\nSave"
    ID_FILE_SAVE_AS        "Save the active document with a new name\nSave As"
    ID_FILE_PAGE_SETUP     "Change the printing options\nPage Setup"
    ID_FILE_PRINT_SETUP    "Change the printer and printing options\nPrint Setup"
    ID_FILE_PRINT          "Print the active document\nPrint"
    ID_FILE_PRINT_PREVIEW  "Display full pages\nPrint Preview"
END


STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT           "Display program information, version number and copyright\nAbout"
    ID_APP_EXIT            "Quit the application; prompts to save documents\nExit"
END


STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1      "Open this document"
    ID_FILE_MRU_FILE2      "Open this document"
    ID_FILE_MRU_FILE3      "Open this document"
    ID_FILE_MRU_FILE4      "Open this document"
END
```

```
STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE          "Switch to the next window pane\nNext Pane"
    ID_PREV_PANE          "Switch back to the previous window pane\nPrevious Pane"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_SPLIT       "Split the active window into panes\nSplit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR         "Erase the selection\nErase"
    ID_EDIT_CLEAR_ALL     "Erase everything\nErase All"
    ID_EDIT_COPY          "Copy the selection and put it on the Clipboard\nCopy"
    ID_EDIT_CUT           "Cut the selection and put it on the Clipboard\nCut"
    ID_EDIT_FIND          "Find the specified text\nFind"
    ID_EDIT_PASTE         "Insert Clipboard contents\nPaste"
    ID_EDIT_REPEAT        "Repeat the last action\nRepeat"
    ID_EDIT_REPLACE       "Replace specific text with different text\nReplace"
    ID_EDIT_SELECT_ALL    "Select the entire document\nSelect All"
    ID_EDIT_UNDO          "Undo the last action\nUndo"
    ID_EDIT_REDO          "Redo the previously undone action\nRedo"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_VIEW_TOOLBAR       "Show or hide the toolbar\nToggle ToolBar"
    ID_VIEW_STATUS_BAR    "Show or hide the status bar\nToggle StatusBar"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE        "Change the window size"
    AFX_IDS_SCMOVE        "Change the window position"
    AFX_IDS_SCMINIMIZE    "Reduce the window to an icon"
    AFX_IDS_SCMAXIMIZE    "Enlarge the window to full size"
    AFX_IDS_SCNEXTWINDOW  "Switch to the next document window"
    AFX_IDS_SCPREVWINDOW  "Switch to the previous document window"
    AFX_IDS_SCCLOSE       "Close the active window and prompts to save the documents"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCRESTORE     "Restore the window to normal size"
    AFX_IDS_SCTASKLIST    "Activate Task List"
```

138

```
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_PREVIEW_CLOSE    "Close print preview mode\nCancel Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_ACCESS_LOGON          "Log on as a registered user"
    ID_ACCESS_LOGOFF         "Quit your user account"
    ID_PREFERENCES           "Set Directories"
END


#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#include "res\Gnd.rc2"  // non-Microsoft Visual C++ edited resources

#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES
#include "afxres.rc"         // Standard components
#include "afxprint.rc"       // printing/print preview resources


/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

Resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by GND.RC
//
#define IDD_ABOUTBOX              100
#define IDR_MAINFRAME             128
#define IDD_USERLOGIN             129
#define IDD_CH_CONTROL            130
#define IDD_CH_FILESYSTEM         131
#define IDD_CH_MAIL               132
#define IDD_CH_OSCONTROL          133
#define IDD_CH_SCRIPTS            134
#define IDD_CH_TASKCONTROL        135
```

```
#define IDD_CH_TELEMETRY              136
#define IDD_CH_MEMORY                 137
#define IDD_MAIN                      140
#define IDD_PREFERENCES               141
#define IDC_LOGIN_LOGIN               500
#define IDC_LOGIN_PASSWORD            501
#define IDC_SYSTEM_TIME               900
#define IDC_STOPWATCH                 901
#define IDC_START_STOPWATCH           902
#define IDC_PAUSE_STOPWATCH           903
#define IDC_SEND                      997
#define IDC_RECEIVE                   998
#define IDC_COMBO_USERLOG             999
#define IDC_LOG_USER                  1000
#define IDC_LOG_MACRO                 1001
#define IDC_LOG_PCL                   1002
#define IDC_MACRO_0                   1050
#define IDC_MACRO_1                   1051
#define IDC_MACRO_2                   1052
#define IDC_MACRO_3                   1053
#define IDC_MACRO_4                   1054
#define IDC_MACRO_5                   1055
#define IDC_MACRO_6                   1056
#define IDC_MACRO_7                   1057
#define IDC_MACRO_8                   1058
#define IDC_MACRO_9                   1059
#define IDC_MACRO_10                  1060
#define IDC_MACRO_11                  1061
#define IDC_MACRO_12                  1062
#define IDC_MACRO_13                  1063
#define IDC_MACRO_14                  1064
#define IDC_MACRO_15                  1065
#define IDC_MACRO_NEXT                1066
#define IDC_EDIT1                     1067
#define IDC_EDIT2                     1068
#define IDC_EDIT3                     1069
#define IDC_EDIT4                     1070
#define IDC_EDIT5                     1071
#define IDC_EDIT6                     1072
#define IDC_EDIT7                     1073
#define IDC_SCRIPT_NORMALEDIT         2000
#define IDC_SCRIPT_EXPRESSEDIT        2001
#define IDC_SCRIPT_SCRIPT             2002
#define IDC_SCRIPT_MACRO              2003
#define IDC_SCRIPT_LISTSCRIPT         2004
#define IDC_SCRIPT_LISTCMD            2005
#define IDC_SCRIPT_EDITCMD            2006
```

```
#define IDC_SCRIPT_P                2007
#define IDC_SCRIPT_ERASELINE        2009
#define IDC_SCRIPT_INSERTEL         2010
#define IDC_SCRIPT_CUTTOEL          2011
#define IDC_SCRIPT_INSERT           2012
#define IDC_SCRIPT_EDIT             2013
#define IDC_SCRIPT_LOAD             2014
#define IDC_SCRIPT_SAVE             2015
#define IDC_SCRIPT_SAVEAS           2016
#define IDC_SCRIPT_NEW              2017
#define IDC_SCRIPT_DELETE           2018
#define IDC_TASK_RADIOAUTO          3000
#define IDC_TASK_RADIOLIST          3001
#define IDC_TASK_RADIOADD           3002
#define IDC_TASK_GRIDTASK           3003
#define IDC_TASK_GRIDFILES          3004
#define IDC_TASK_EDITFILE           3005
#define IDC_TASK_ADD                3006
#define IDC_TASK_GETLIST            3007
#define IDC_TASK_DELETE             3008
#define IDC_TASK_LOAD               3009
#define IDC_MAIL_FROM               4000
#define IDC_MAIL_TO                 4001
#define IDC_MAIL_TIME               4002
#define IDC_MAIL_SUBJECT            4003
#define IDC_MAIL_MAILFILENAME       4004
#define IDC_MAIL_LISTMAILFILE       4005
#define IDC_MAIL_LISTMAIL           4006
#define IDC_MAIL_GETDIR             4007
#define IDC_MAIL_READMSG            4008
#define IDC_MAIL_ADDMSG             4009
#define IDC_MAIL_DELMSG             4010
#define IDC_MAIL_PURGEMSG           4011
#define IDC_FILE_LISTFILE           5001
#define IDC_FILE_GETDIR             5002
#define IDC_FILE_READFILE           5003
#define IDC_FILE_ADDFILE            5004
#define IDC_FILE_DELFILE            5005
#define IDC_FILE_PURGEFILE          5006
#define IDC_FILE_SELECTFILE         5007
#define IDC_FILE_SELECTCOMBO        5008
#define ID_ACCESS_LOGON             32771
#define ID_ACCESS_LOGOFF            32772
#define ID_PREFERENCES              32773
#define ID_DEBUG_LOADMACRO          32774

// Next default values for new objects
```

```
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS              1
#define _APS_NEXT_RESOURCE_VALUE      139
#define _APS_NEXT_COMMAND_VALUE       32776
#define _APS_NEXT_CONTROL_VALUE       5000
#define _APS_NEXT_SYMED_VALUE         101
#endif
#endif
```

Password.h

```
// password.h : header file
//


/////////////////////////////////////////////////////////////////////////////
// CPasswordDlg dialog

class CPasswordDlg : public CDialog
{
// Construction
public:
        CPasswordDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CPasswordDlg)
        enum { IDD = IDD_USERLOGIN };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA



// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPasswordDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CPasswordDlg)
                // NOTE: the ClassWizard will add member functions here
        //}}AFX_MSG
```

```
                DECLARE_MESSAGE_MAP()
};
```

Password.cpp

```cpp
// password.cpp : implementation file
//

#include "stdafx.h"
#include "gnd.h"
#include "password.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPasswordDlg dialog


CPasswordDlg::CPasswordDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CPasswordDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CPasswordDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}



void CPasswordDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CPasswordDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}



BEGIN_MESSAGE_MAP(CPasswordDlg, CDialog)
        //{{AFX_MSG_MAP(CPasswordDlg)
                // NOTE: the ClassWizard will add message map macros here
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

PrefDlg.h

```cpp
// prefdlg.h : header file
//


//////////////////////////////////////////////////////////////////////////
// CPrefDlg dialog

class CPrefDlg : public CDialog
{
// Construction
public:
        CPrefDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        struct PANSATFileInfo (*pOldPFI)[MAXDIRS], NewPFI[MAXDIRS];
        BOOL m_bHasChanged;

        //{{AFX_DATA(CPrefDlg)
        enum { IDD = IDD_PREFERENCES };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPrefDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CPrefDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnChangeEdit();
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

PrefDlg.cpp

```
// prefdlg.cpp : implementation file
//

#include "stdafx.h"
#include "gnd.h"
#include "prefdlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
// CPrefDlg dialog


CPrefDlg::CPrefDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CPrefDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CPrefDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CPrefDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CPrefDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPrefDlg, CDialog)
        //{{AFX_MSG_MAP(CPrefDlg)
        ON_EN_CHANGE(IDC_EDIT1, OnChangeEdit)
        ON_EN_CHANGE(IDC_EDIT2, OnChangeEdit)
        ON_EN_CHANGE(IDC_EDIT3, OnChangeEdit)
        ON_EN_CHANGE(IDC_EDIT4, OnChangeEdit)
        ON_EN_CHANGE(IDC_EDIT5, OnChangeEdit)
        ON_EN_CHANGE(IDC_EDIT6, OnChangeEdit)
        ON_EN_CHANGE(IDC_EDIT7, OnChangeEdit)
```

```
        //}}}AFX_MSG_MAP
END_MESSAGE_MAP()


////////////////////////////////////////////////////////////////////////
// CPrefDlg message handlers

BOOL CPrefDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        int i;
        CEdit *pEdit;

        for (i=0; i<MAXDIRS; i++)
        {
          pEdit = (CEdit *)GetDlgItem(prefID[i]);
          pEdit->SetWindowText((*pOldPFI)[i].Dir);
        }

        m_bHasChanged = FALSE;
        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}


void CPrefDlg::OnChangeEdit()
{
  m_bHasChanged = TRUE;
}


void CPrefDlg::OnOK()
{
        int i;
        CEdit *pEdit;

        for (i=0; i<MAXDIRS; i++)
        {
          pEdit = (CEdit *)GetDlgItem(prefID[i]);
          pEdit->GetWindowText(NewPFI[i].Dir);
        }

        CDialog::OnOK();
}
```

IX.   INITIAL DISTRIBUTION LIST