TS3010 ANTENNA TRACKING SYSTEM

CMU SOFTWARE PRODUCT DESCRIPTION

Author: DJL Brown

$Date:: 15 Jul 1999$

CONTENTS **Page**

1. AMENDMENTS....5
2. INTRODUCTION....7
3. ASSOCIATED DOCUMENTS....7
4. NAMING CONVENTIONS....8
5. APPLICATION GENERAL DESCRIPTION....9
5.1. Function within TS3010....9
5.2. Overall Structure....9
5.3. Multiple Instances....10
5.4. External Interactions....10
5.4.1. Inter-Computer Data Exchanges....10
5.5. Structural Philosophy....11
5.5.1. Display Control....11
5.5.2. Data Storage....11
5.5.3. Threads....11
5.5.4. Inter-Thread Communications....11
5.6. Special Considerations....12
5.6.1. Display Updating....12
5.6.2. Use of the MFC Messaging System....12
5.6.3. Modal vs. Non-Modal Dialog Boxes....12
5.6.4. Use of Afx Message Boxes....12
6. CLASS OBJECT RELATIONSHIPS....13
6.1. Inheritance....13
6.2. Dependency and Ownership....15
6.3. Friends and Globally Visible Functions....15
7. OBJECT RESPONSIBILITIES....16
7.1. The Application....16
7.1.1. Function....16
7.1.2. Dependants....16
7.2. The Main Frame....16
7.2.1. Function....16
7.2.2. Dependants....16
7.3. The Document....17
7.3.1. Function....17
7.3.2. Dependants....17
7.4. The Condition Panel....17
7.4.1. Function....17
7.4.2. Dependants....18
7.5. The Control Panel....18
7.5.1. Function....18
7.5.2. Dependants....18

(Cont…

CONTENTS (Cont.) **Page**

7.6. Form Views 18
7.6.1. Function 19
7.6.2. Common Facilities 19
7.6.3. Positioning and Tracking 20
7.6.3.1. Function 20
7.6.3.2. Dependants 20
7.6.4. Trending 21
7.6.4.1. Function 21
7.6.4.2. Dependants 22
7.6.5. Beacon and Polariser Control 23
7.6.5.1. Function 23
7.6.5.2. Dependants 23
7.6.6. System Status 23
7.6.6.1. Function 23
7.6.6.2. Dependants 24
7.7. The System Configuration Dialog Box 24
7.7.1. Function 24
7.7.2. Dependants 24
7.8. The Set System Time Dialog Box 24
7.8.1. Function 24
7.8.2. Dependants 24
7.9. The Stowing Functions Dialog Box 24
7.9.1. Function 24
7.9.2. Dependants 25
7.10. Alarm and Event Display 25
7.10.1. Function 25
7.10.2. Dependants 25
7.11. The Editor 25
7.11.1. Function 26
7.11.2. Dependants 26
8. DATA EXCHANGE MESSAGE HANDLING 27
8.1. Satellite-specific Default Settings 27
9. INSTALLATION-SPECIFIC REQUIREMENTS 30
9.1. Isolation of Unique Code 30
9.2. Antenna DLL Requirements 30
9.2.1. Library Type 30
9.2.2. Required Functions 30
9.2.2.1. Construction Function 30
9.2.2.2. Dll Version Function 31
9.2.2.3. Satellite Data Beacon Page Function 31
9.2.2.4. PLC Serial Channel Function 32
9.2.3. Form View Requirements 32
9.2.4. Property Page Requirements 33
9.2.5. PLC Serial Channel Requirements 34
9.2.6. Required Resources 34

(Cont…

CONTENTS (Cont.) **Page**

9.3. Polariser Dll Requirements 35
9.3.1. Library Type 35
9.3.2. Library Name 35
9.3.3. Required Functions 35
9.3.3.1. Construction Function 35
9.3.3.2. Dll Version Function 35
9.3.3.3. Satellite Data Polariser Page Function 36
9.3.4. Dialog Box Requirements 36
9.3.5. Property Page Requirements 36
9.3.6. Required Resources 37
9.4. Help System Additions 37

1. AMENDMENTS

DATE	DESCRIPTION	AUTHORITY	AMEND. No.

THIS PAGE IS INTENTIONALLY BLANK

2. INTRODUCTION

2.1. The TS3010 Control and Monitoring Unit (CMU) is a Windows application. It is designed to run under Microsoft Windows NT 4, on an Intel Pentium processor or equivalent. At first release, the Windows operating system used was Windows NT4.0 (Service Pack 3).

2.2. The CMU application has been built using Microsoft's Visual C++ Version 5 (VC), in Microsoft's Developer Studio (as supplied with Visual C++ Version 5).

2.3. Extensive use has been made of the Microsoft Foundation Class (MFC) library, on which most class objects in the CMU code are based.

2.4. Being based on MFC, the CMU Application source code is built around the concept of a class object *wrapping* a Window object. That is, the code in general does not deal directly with window objects, but with class objects which hide the window objects from the programmer. In reading this document, a very clear distinction must be maintained between the two groups. An attempt to maintain the difference has been made by generally referring to the window objects as purely *windows,* whereas the wrapping classes are referred to as *window objects* or *window classes.* However, for convenience when discussing the program, references are often made to a window's name on its own when in fact the window-wrapping class is meant (eg a reference to 'Main Frame' would often be used as short-hand for 'the class object which wraps the main frame window'). Further, names used in descriptive text (both here and in comments embedded in the source code) are generally a 'plain English' version of the actual name used within the code. The 'Main Frame' example is a case in point, actually referring to a class titled CMainFrame.

3. ASSOCIATED DOCUMENTS

3.1. The reader is referred to the following documents, for a detailed insight into indicated aspects of the CMU software:-

a. TUNRA DOC 173 – CMU Serial Communications Software;
b. TUNRA DOC 174 – CMU Inter-Thread Communications, and
c. TUNRA DOC 175 – CMU Error Handling System.

3.2. It should be noted that since these documents were written, some aspects of the systems involved have been modified. However the documents remain an accurate reflection of the method of operation of the systems discussed therein. The changes which have been made effect detail only, and are evident from the source code comments

4. NAMING CONVENTIONS

4.1. Microsoft's Visual C++ (VC) uses a number of conventions in naming objects. Most of these have been adhered to in the development of CMU, with some minor variations. One of these, found to be particularly useful, is to prefix the name of classes defined within the CMU source code with 'cl' - VC uses 'C'. This variation makes differentiation between CMU classes and VC classes instant and easy.

4.2. As in VC, a form of Hungarian notation is used in naming variables. Some minor variations between VC and the CMU code however exist (developer's ego!). The most common of these are given in Table 1. This is not by any means a complete list, but programmers should have no difficulty in recognising other, unlisted variations. (Note however that even within the CMU code, some minor variations exist. The code relating to serial data, for instance, uses 'C' for a class prefix rather than 'cl'.)

Table 1: Hungarian Notation

Object	VC Prefix	CMU Prefix
Boolean	b	bo
Integer	n	in
Unsigned integer	u	uin
double	dbl	df
Character array	None	ch
String (NULL-terminated)	None	sz
CString object	None	cs
Void	None	vd
Function	None	f
Structure	(None)	st
Class	C	cl

4.3. A convention rigidly adhered to is that all members of a class are prefixed with 'm_', whether they be variable, class, structure or function.

4.4. CMU prefixes may be cascaded - for instance a function which returns an integer and is a class member would be prefixed 'm_fin'.

4.5. Generally, underlines ('underscores' in American English, and now –unfortunately, in the writer's view {we no longer write on slates!!} - almost universal in programming parlance) in naming are avoided, and capitalisation of the first letter of words is used to indicate word breaks.

5. APPLICATION GENERAL DESCRIPTION

5.1. Function within TS3010

5.1.1. The CMU provides the operational interface with TS3010. It is the only part of TS3010 which normally has a display and keyboard (These items are added to the Antenna Tracking and Control (ATCU) for maintenance, repair and installation purposes only).

5.1.2. Whilst all antenna control parameters (e.g. position control loop settings, software limits) and tracked object parameters (e.g. satellite-specific settings, Intelsat data) are stored in the ATCU computer, the CMU stores operational information such as alarm logs, Trending data, last-used display settings.

5.1.3. The CMU application is designed to be maintained as a universal entity. That is, it should configure itself to match any particular installation.

5.2. Overall Structure

5.2.1. The CMU is a *dynamically-linked* application. Copies of Microsoft Foundation Class (MFC) dynamic link libraries (dlls) are installed with the CMU executable (CMU.exe), and are automatically linked to CMU.exe at run-time. These libraries provide CMU.exe with the MFC code necessary for it to operate.

5.2.2. There is no code within CMU.exe that is specific to an antenna or polariser. All such code is provided by additional dlls.

5.2.3. A dll has been built for each type of antenna installation, and for the polariser currently in use. For uniformity, the CMU setup installs all antenna and polariser dlls so far built, even though only the dll for the connected antenna (and possibly the dll for the polariser) are actually used. Thus any CMU computer may operate on any antenna for which a dll has been provided.

5.2.4. The sequence of events relating to the connection of dlls is as follows:-

a. At start-up, CMU.exe attaches the MFC dlls to itself;
b. on receipt of an ATCU Regular Poll message, the CMU queries the ATCU for antenna-specific information. Included in the returned message is an Antenna Type Number;
c. the CMU builds an antenna dll file name from the supplied Antenna Type Number. The dll file name is of the form

AntennaTypeN.dll

where N is a number (which may have any number of digits);
d. CMU.exe searches its default directory for a file with the required name. If it exists, CMU.exe attaches the dll file to itself, and executes a special function provided by the dll. That function may search for and attach a polariser dll.

5.3. Multiple Instances

5.3.1. The CMU application is capable of existing as multiple instances running on the same computer. As each instance will be connected to a different ATCU, the instances must be totally isolated from one another.

5.3.2. When an instance of CMU is created, one of the first things it does is to determine how many instances already exist. This information is used in accessing information stored in the Windows NT Registry - for instance to which serial port(s) it should try to connect, and what the last Trend chart settings used were.

5.3.3. The maximum number of concurrent instances of CMU is limited only by the serial ports available, and the capacity of the computer to run them.

5.4. External Interactions

5.4.1. The CMU communicates with the ATCU (and PLC, if one is included in the installation). Where two CMUs are in use, they both communicate with the ATCU (and PLC), the ATCU providing each with information about the other.

5.4.2. Two CMUs on one system do not directly interact with each other. Any apparent interaction is actually performed by the ATCU.

5.4.3. CMU/ATCU communication is via a serial data link. The hardware format of the link may be either RS323 or RS422. The software is not configured to handle an RS485 link.

5.4.4. The CMU and ATCU software determine the data link settings. The two programs must therefore include the same settings, which are:-

Baud Rate:	9600
Data Bits:	8
Stop Bits:	1
Parity:	None
Flow Control:	None

The same settings are used for the CMU/PLC data links, when a PLC is used.

5.4.1. Inter-Computer Data Exchanges

5.4.1.1. Once every second, the ATCU transmits (on both its CMU data links) a *Regular Poll*. This message contains all of the system data needed by a CMU to display the current condition of the TS3010 system, the antenna and its Drive system.

5.4.1.2. The Regular Poll may vary slightly with the installation. This variation is generally restricted to the length of a set of data bytes which appear at the end of the message.

5.4.1.3. A CMU may send polls to the ATCU. These may be commands to do something, or requests for data.

5.4.1.4. The first poll that a CMU sends to the ATCU after it receives the first Regular Poll following connection, is a request for installation-specific information. When it receives the ATCU's response to this, the CMU uses the information to configure itself to match the installation.

5.5. Structural Philosophy

5.5.1. The CMU Application is fundamentally an *event-responsive* program. It does nothing unless it receives input from the user or the ATCU (or PLC, if one is used).

5.5.1. Display Control

5.5.1.1. As it is based on the MFC system, the CMU consists largely of Windows *structures* wrapped in Windows *class objects*. The details of the windows structures are hidden from the programmer, who manipulates them via *member functions* in the wrapping class objects. Each Window structure however is based on a *resource* which is built by drawing the required window's appearance in the Developer Studio's resource editor.

5.5.1.2. A 'Main Frame' window contains two dialog bars which are always in view. One of these displays basic system information (the 'Condition' bar), the other a few basic controls(the 'Control' bar). These two bars are anchored to the Main Frame. Between them, one of four 'Form-View' windows is displayed. (The operator normally makes the selection, via the Main Frame's 'View' menu.) Data not accessed directly via these displays is made available through a variety of dialog boxes.

5.5.2. Data Storage

5.5.2.1. Unlike most Windows programs, the CMU does not have a 'View' window, in the sense used in MFC - usually a text display and editing medium. It does have a 'Document' class object (normally associated with a 'View'), but this is used to manipulate information rather than to store it.

5.5.2.2. Although not often recognised as 'good practice' by C programmers, the use of global data structures and variables has been found convenient in this Application, due to the fact that a large amount of data being handled is used by a wide range of entities within the program. A small collection of simple utility functions, for general use, are included with the global data. (All of the global items are *defined* in one header file, which defines nothing else. They are *declared* in one file, which declares nothing else.)

5.5.3. Threads

5.5.3.1. All top-level activity occurs in one thread - the *User Interface Thread*. This thread handles all operator interactions, display control and data manipulation.

a) Each serial data link in use (one for the ATCU link, a second for a PLC link) is operated through two dedicated *Worker Threads* - one for transmission, one for reception. Worker threads do not interact with the operator or display, nor do they encode/decode data within messages. They do handle serial link message protocols, and facilitate the dispatch and receipt of messages.

5.5.4. Inter-Thread Communications

5.5.4.1. The User Interface Thread communicates with each Worker Thread via a mail-box system. The mail-boxes (one for messages to be transmitted, one for received messages) associated with a worker thread are built when the thread is created.

5.5.4.2. The Worker Threads do not communicate with each other.

5.6. Special Considerations

5.6.1. The function of TS3010 is fundamentally a *real-time* operation. Thus, unlike most Windows applications, CMU must take account of 'simultaneous' occurrences and appear to respond in real time. The most significant implication of this fact occurs in the presentation of basic system information, such as time, signal strength, antenna position etc.

5.6.1. Display Updating

5.6.1.1. All the information displayed to the operator must be updated whenever that information changes, regardless of what the operator is doing. Thus it is not permissible, for instance, for system status information to be frozen while a dialog box is visible. This fact requires careful consideration of how dialog boxes are created, and how the whole display system is updated. The way in which this is achieved is as follows:-

a) Receipt of any data from the ATCU (or PLC) results in Document calling a member function of Main Frame.
b) The Main Frame function calls member functions of both Dialog Bars, the Frame View currently visible and any dialog box owned by Main Frame which is visible.
c) Each Dialog Bar, Frame View and dialog box member function called updates its window according to the data it displays.

5.6.2. Use of the MFC Messaging System

5.6.2.1. It should be noted that the use of MFC's messaging system to carry out display updating is avoided. If MFC messaging were to be used, it would be interrupted by, for instance, the display of modal dialog boxes and use of the Help system, resulting in a display freeze during those occurrences.

5.6.2.2. The MFC messaging system is however used to facilitate data and command flow to and from displayed windows' class objects. The display update requirement does not prevent this, as such exchanges are single events and virtually instantaneous.

5.6.3. Modal vs. Non-Modal Dialog Boxes

5.6.3.1. When designing a dialog box, the functional requirement of what the operator may need to do, and what, if anything, he should be prevented from doing while it is displayed, must be born in mind. If, for instance, stowing operations are under way (the Stowing dialog box is displayed) the operator still needs access to controls outside the dialog box. A modal dialog box would prevent this, as modal dialog boxes freeze the User Interface Thread, preventing access to anything outside the dialog box. Thus in this instance, a non-modal dialog box is in order. On the other hand, when accessing satellite data, the operator should be prevented from accessing controls outside the satellite data dialog box. Thus the use of a modal box is acceptable.

5.6.4. Use of Afx Message Boxes

5.6.4.1. MFC provides an apparently very convenient, ready-made dialog box for general message display - AfxMessageBox. This box acts like a modal dialog box, but it does not in fact interrupt the User Interface thread. Thus particular care must be taken with its use, as it is very easy to get into a recursive calling situation, resulting in a total system lock-up. Another disadvantage of AfxMessageBox is that its layout and colour are not controllable from within the Application.

5.6.4.2. To bypass the disadvantages of AfxMessageBox, a general message dialog box has been provided within CMU. finMessageBox provides the same facilities as AfxMessageBox, with the added capability of being able to generate either a modal or non-modal dialog box. Further, its colour is controllable, and being based purely on the MFC CDialog class, its characteristics are better understood.

6. CLASS OBJECT RELATIONSHIPS

6.1. Inheritance

6.1.1. Most class objects within CMU are derived directly or indirectly from MFC classes. Within the classes associated with windows, there are a few basic types required in CMU which have more capability than the equivalent MFC classes. The class hierachy within CMU for windows objects is shown in Table 2, and for those classes not directly associated with windows, see Table 3.

Table 2: CMU Window Object Class Heirarchy

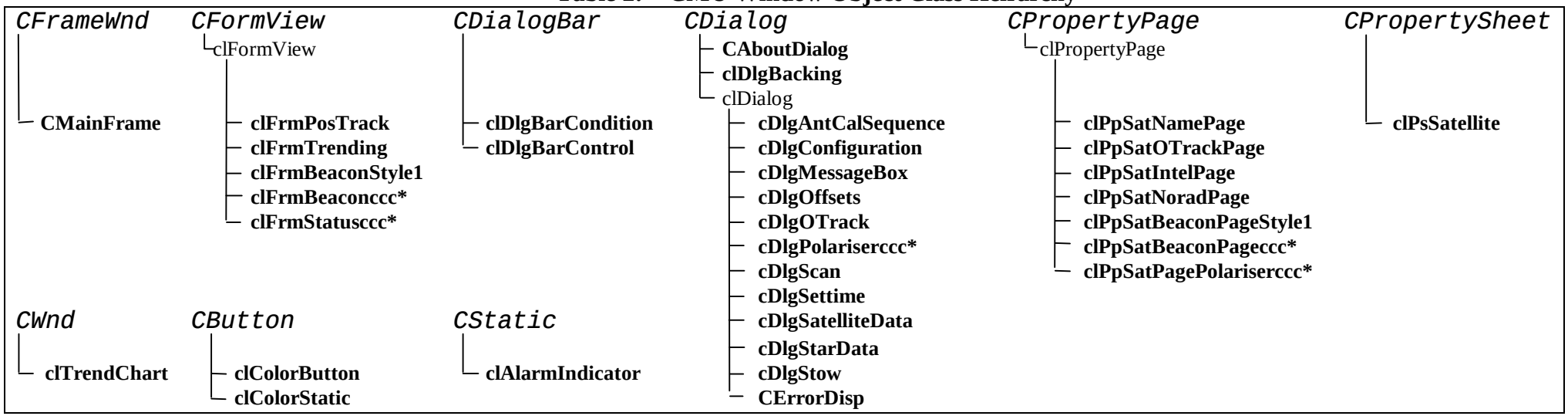

Table 3: CMU Non -Window Object Class Hierachy

Notes:

1. MFC classes are shown in italics.
2. CMU Classes which wrap visible objects are shown in bold type.
3. The letters ‘ccc’ included in a class name indicates that several classes with similar function, but with detail dependent upon the installation, are represented. The ‘ccc’ will indicate the antenna or polariser to which the class relates. (All these class objects are provided by dlls unique to the installation.)
4. A superscript * indicates that this class object is provided by an installation-specific dll.

6.2. Dependency and Ownership

6.2.1. Windows are sometimes children of other windows, and the parent of all windows is the Main Frame. However, the parent/child relationship is not always obvious. This is particularly true of the case of the buttons controls seen inside dialog bars. These control windows are actually children of Main Frame, NOT the dialog bar in which they are located. The notable effect of this is that the MFC class CCmdUI has to be used to update these controls, via message handler functions called when the system issues the appropriate messages. Controls within normal dialog boxes however are children of the window in which they appear, and may be updated through that window's wrapper class' DoDataExchange() function.

6.3. Friends and Globally Visible Functions

6.3.1. Functions within CMU which are not members of a class are used where class members would not be appropriate, or in some cases, even possible.

6.3.2. The alarm/event handling system uses friends of the CError class extensively to perform operations relating to the *collection* of CError class objects which represent current alarms/events. This arrangement is used so that the functions can have access to member variables of the CError class.

6.3.3. The Trending system uses a group of functions not contained within a class to handle the creation, reading and writing of the Trending files. This is simply a matter of convenience in code-writing. The functions are not friends of any class, as the information they need is either contained within them or passed to them as parameters.

6.3.4. A few globally visible functions are provided for convenience. These relate to reading and writing Registry entries, and testing for alarm/event conditions in data received from the ATCU or PLC. As this operation is required to be performed in a number of places - all within the User Interface thread - global functions proved to be the convenient and code-saving means of achieving the requirement.

7. OBJECT RESPONSIBILITIES

7.1. Each major object within CMU is discussed. The list is restricted to those objects where some illumination of function and inter-relationship may be useful. The discussion of objects with self-evident functions is restricted to a mention in the discussion of their parent or owner.

7.1. The Application

7.1.1. The class object that embodies the CMU application is CCMUApp. It is based on the MFC class CWinApp.

7.1.1. Function

7.1.1.1. The functions of CCMUApp are:-

- ? To initialise the instance of CMU and connect it with the system;
- ? To determine which instance this instance is;
- ? To initialise and show the Main Frame window, and to create and attach a CDocument object to it;
- ? To handle 'data received' messages received from the serial communications worker threads receiving data from ATCU and PLC;
- ? To set and reset system resources (eg colours) as required, and
- ? To manage the creation and termination the CMUAlarmBroadcast application;

7.1.2. Dependants

7.1.2.1. CCMUApp, not being a window, does not have children in the Windows sense. However, it does give birth to the following entities:-

- ? CMainFrame (see 'The Main Frame');
- ? CCMUDoc (see 'The Document');
- ? CAboutDialog (wraps the 'About' dialog box), and
- ? Serial communications worker threads represented by their classes
  clATCUChannel and
  clPLCChannel (if required).

7.2. The Main Frame

7.2.1. The Main Frame of the application is wrapped by the class object CMainFrame, which is based on the MFC class CFrameWnd.

7.2.1. Function

7.2.1.1. CMainFrame controls the display of the application's menus, dialog bars and views. It is responsible for bringing into being the application's two dialog bars and locating them. It is also responsible for creating the form views which fill the space between the two dialog bars, as required and instructed by other parts of the application.

7.2.2. Dependants

7.2.2.1. The direct children of the Main Frame window are represented by the following class objects:-

- ? clDlgBarCondition (see 'The Condition Panel');
- ? clDlgBarControl (see 'The Control Panel');

? All classes based on clFormView (see 'Form Views').

7.2.2.2. Main Frame owns a number of pop-up windows, which are generated in response to menu demands. These are:-

? clDlgConfiguration (see 'The Configuration Dialog Box');
? clDlgSettime (wraps the Set-Time Dialog Box), and
? CErrorDisp (see 'Alarm and Event Display').

7.3. The Document

7.3.1. The Document is embodied entirely in the functions and variables which are members of the CCMUDoc class object, created by CCMUApp.

7.3.1. Function

7.3.1.1. The Document acts as a clearing centre for data and commands flowing to and from the ATCU (and PLC). Its functions are:-

? To initialise globally visible arrays, structures and variables which hold installation-specific and data and data for general use;
? To analyse, verify and decode data received from ATCU and PLC;
? To initiate display updates when new data is received (which it does by calling a CMainFrame member function), and
? To regulate the sending of commands and requests to ATCU and PLC, and to track the response to these commands and requests.

7.3.2. Dependants

7.3.2.1. Document has no dependants.

7.4. The Condition Panel

7.4.1. The Condition Panel always fills the top quarter of the full-size CMU display. It is always visible if the CMU display is large enough to house it and the Control Panel.

7.4.1. Function

7.4.1.1. The Condition Panel displays major system data in real time. The data displayed includes:-

? System Time and Date;
? source of System Time and Date;
? beacon signal strength;
? antenna identity;
? ATCU 'State' and operating 'Mode';
? the set satellite (or star, if in StarTrack Mode);
? antenna pointing angles (azimuth and elevation), and
? system Control Point.

7.4.1.2. A further function of the Condition Panel is to provide the facility of changing the System Control Point.

7.4.2. Dependants

7.4.2.1. The display of each item of data is achieved through a child window wrapped by an MFC CStatic class object.

7.4.2.2. A button control provides the means of changing the System Control Point. This control is an 'owner-drawn' button, the face colour and 'name' of which changes to reflect the current point of control. It is disabled if a change of System Control Point from the CMU is not allowed. Although this button is a *child* of the Condition Panel, it is actually *owned* by the Main Frame. Therefore it has to be updated via the rather obscure mechanism used by MS Windows to update all Main Frame controls (which in most Windows applications are restricted to menus and buttons directly associated with menu items).

7.5. The Control Panel

7.5.1. Forming a small strip across the bottom of the CMU display, the Control Panel is the last window in the CMU display to be clipped as the vertical size of the display is reduced.

7.5.1. Function

7.5.1.1. The Control Panel contains four button controls. The two centre buttons allow for major antenna Drive operations. A button on the right provides access to Stowing operations, the left-hand button returning the centre window display to the previous window.

7.5.2. Dependants

7.5.2.1. The dependants of Control Panel are:-

- ? Drive Start/Stop control button;
- ? Drive Start Progress indicator;
- ? Antenna Go/Hold control button;
- ? Stowing button;
- ? Stowing Operations dialog box, and
- ? 'Go Back' control button.

7.5.2.2. Although the button controls are *children* of the Control Panel, they are actually *owned* by the Main Frame, and therefore are updated via the mechanism used by MS Windows to update all Main Frame controls.

7.5.2.3. The Drive and Antenna control buttons are owner-drawn, with face colours to reflect the current applicable condition. Their 'names', as shown on their faces, toggle with the current applicable condition.

7.5.2.4. The Drive Start Progress indicator is hidden unless a Drive Start sequence is in progress. Its timing is set from information obtained from the ATCU when the CMU instance first connects to an ATCU, and its only function is to provide the operator with a rough indication that something is happening.

7.6. Form Views

7.6.1. The space between the Condition Panel and the Control Panel is always filled by a *Form View.* The form view displayed is selected by the operator via Main Frame's *Views* menu. The possible displays are:-

- ? Positioning and Tracking;
- ? Trending;
- ? Beacon Setup, and
- ? System Status.

The last two views listed are unique to the connected antenna/ATCU. Their window resources and wrapper class objects are provided from within an antenna-specific dll. The dll is selected and attached when the CMU receives identifying information from the ATCU.

7.6.2. All the form views are wrapped in class objects based on clFormView, a class derived from the MFC CFormView class.

7.6.1. Function

7.6.1.1. The form views provide access to, and information about, a wide range of variables and settings within the TS3010 system.

7.6.2. Common Facilities

7.6.2.1. the base class clFormView provides functionality to each form view in addition to that afforded by its base - CFromView - as follows:-

a. Sets the background of the formview, its static controls and inactive edit controls, to a colour nominated by a defined constant;
b. stops transfer of data to/from the window unless the window is visible, and
c. provides basic control on leaving a CEdit control, via the following member variables and overridable functions:-

m_boUpdateAllowed;
m_boEditComplete;
m_boDataError;
m_inLastChangedEditBoxID;
m_inNewFocusControlID;
OnChangeEdit()
OnLostFocus()
OnOk()

Each form view resource is provided with a hidden OK button, so that any Enter key-stroke associated with any edit control in the displayed form view will result in the wrapper class' OnOk() function being called.

7.6.2.2. By mapping the Windows messages ON_EN_CHANGE and ON_LOSTFOCUS, and calling the OnOk() function when Enter is pressed, control of any editing operation is implemented thus:-

a. If a change is made in an edit control, and the focus is moved away from it without Enter being pressed, a message box is generated asking if the change should be kept. Clicking the YES button completes the process as if Enter had been pressed. Clicking the NO button returns the control's entry to the original value.

b. As soon as a change is made in an edit control, update of the form view is inhibited. If this were not done, the regular display updates would make editing changes impossible. Normal display updating will be inhibited (in the connected window) until the clFormView class member m_boUpdateAllowed is set when the new setting has been processed by the OnOk() function.
c. If an entry error is detected (by the MFC function DoDataExchange()) the operator is notified via a dialog box generated by the MFC data exchange operation, and clFormView's member m_boDataError is set, allowing the derived class to take any required action - such as avoiding sending the erroneous data to the ATCU. On dismissal of the advising dialog box, the focus is returned to the control containing the offending entry.

7.6.3. Positioning and Tracking

7.6.3.1. The Positioning and Tracking form view window is wrapped by the clFrmPosTrack class object, which is derived from clFormView.

7.6.3.1. Function

7.6.3.1.1. Provides facilities for:-

- ? Setting antenna position or rate commands;
- ? selecting ATCU operating Mode;
- ? making setting changes associated with the selected ATCU Mode;
- ? selecting the satellite or star to be used, and
- ? reviewing and changing default data associated with each selectable satellite or star.

This form view also displays the Software Limits currently applied to antenna motion.

7.6.3.2. Dependants

7.6.3.2.1. The centre of the Positioning and Tracking window is occupied by a child window selected according to the ATCU Mode set, from the following list:-

a. Offsets - wrapped by the class clDlgOffsets. Displayed when Position Mode, Intelsat Mode or StarTrack Mode is selected. Allows 'nudging' of the antenna while following a command or model;
b. OrbitTrack Control - wrapped by clDlgOTrack, this window allows changing between OrbitTrack Start types. It is displayed only when the OrbitTrack Mode is selected;
c. Scan Settings - wrapped by clDlgScan, this window is displayed when Scanning Mode is set. It provides access to the settings used during a Scan operation.

7.6.3.2.2. Another child window is displayed at right-centre of this form view when StarTrack Mode is set. This window is wrapped by the class clDlgAntCalSequence. It allows the operator to set parameters for the in-built Calibration Sequence, and to control the operation of that sequence. (This sequence may be used to advantage when calibrating pointing errors and receiver performance during star tracking.) The window contains a number of standard Edit controls and two standard button controls.

7.6.3.2.3. A standard MFC Combo Box control is used to provide for ATCU Mode selection. This is configured to operate with a 'drop-down' list, which is pre-programmed with the names of all ATCU Modes.

7.6.3.2.4. Access to satellite and star data is provided via a special 'combo box' style control, the operating code for which is included in the class which wraps the Position and Tracking form view - clFrmPosTrack. Although the control appears to operate as a standard MFC Combo Box control, it has three features not realizable by use of the MFC standard control:-

a. A 'right-click' on an item in the dropped list generates a pop-up dialog box which provides access to that item's data;
b. a small panel appears at the head of the dropped list advising of the above, and
c. a 'backing' window is provided behind the dropped list. This prevents the Calibration Sequence window over-painting the dropped list when in StarTrack Mode. (This complication is necessary because both the Calibration Sequence window and the dropped list of the 'combo box' are children of the form view window, each with equal status. Control of the dropped list of an MFC combo box is well hidden from the programmer, and it cannot be forced to 'stay on top'. Due to area constraints, the dropped list and the Calibration Sequence windows must overlap.)

7.6.3.2.5. The Satellite Data dialog box is a complex marriage of dialog box, property sheet and property pages. The dialog box is the container. It allows display of the name of the satellite (in its title bar) to which all the information within it applies. A child property sheet 'fills' the dialog box client window, and this is parent to a number of property pages. Each property page contains a number of standard edit controls, plus a 'Set' button. No action is taken on an edit action until the Set button is activated. If another page is selected or an attempt made to close the dialog box without activating the Set button, a warning is given (in similar manner to that provided in the form views) and the operator is allowed to accept or discard the change made. This facility, together with entry error handling, is provided by the base class of all the property pages, clPropertyPage – itself based on the MFC class CPropertyPage. The operations are similar to those described for form view objects.

7.6.3.2.6. The Star Data dialog box is a simple dialog box containing several standard edit controls and a Set button. If an attempt is made to close the dialog box without activating the Set button, a warning is given and the operator is allowed to accept or discard the change made. . This facility, together with entry error handling, is provided by the base class of all the dialog boxes containing controls, clDialog – itself based on the MFC class CDialog. The operations are similar to those described for form view objects.

7.6.4. Trending

7.6.4.1. The Trending form view window is wrapped by the clFrmTrending class object, which is derived from clFormView.

7.6.4.1. Function

7.6.4.1.1. Provides controls for the operator to manipulate and interpret the Trending chart display. These include:-

- ? Vertical axis scale settings for each chart trace;
- ? Vertical axis offsets for each chart trace;
- ? Selection of Real Time or Historical chart display mode.

7.6.4.1.2. When the chart is in Historical display mode, additional facilities provided are:-

- ? Horizontal axis controls when the chart is in Historical mode:-

a. Chart width;
b. Forward and backward pan controls (half-chart width and 24 hours);
c. Set right-hand edge of the chart to current time;
d. Set cursor position.

? Display of trace readings at cursor position;
? Capability to send cursor position values of azimuth and elevation angles to Positioning and Tracking, as new position demand settings;
? Selection of a section of the chart for listing plotted values in a file, and reviewing the file in SuperEditor (a text editor provided with CMU).

7.6.4.1.3. clFrmTrending does not include the Trending chart itself, which is provided by a child window.

7.6.4.2. Dependants

7.6.4.2.1. The Trending chart window is a child of the Trending form view window. It is created and sized by clFrmTrending during the latter's initialisation, and is wrapped by the clTrendChart class object.

7.6.4.2.2. clTrendChart includes a buffer structure array which holds the variables displayed by it. This array has 300 elements, one for each display point across the horizontal axis. (If the display resolution is 1024x768, two pixels are used for each point displayed, for a chart width of 600 pixels. When the resolution is 800x600, only one pixel is used, and the chart width is 300 pixels.)

7.6.4.2.3. The chart buffer is treated as a circular buffer, and is accessed via 'get' and 'put' pointers.

7.6.4.2.4. In Real-Time mode, values for each trace, and the corresponding time, are added to the buffer each second, after the ATCU Regular Poll has been interpreted by Document. On chart creation, or selection of Real Time mode, the buffer is first loaded with the any data available from the Trending files, covering the 5 minutes (the width of the chart in Real Time mode) prior to the current time.

7.6.4.2.5. In Historical mode, the buffer is fully loaded whenever the Historical mode is selected or a horizontal axis setting is changed. For this purpose, data is extracted from the Trending files.

7.6.4.2.6. After it has interpreted each ATCU Regular Poll (once per second), Document adds time, axis position and beacon strength data to a Trending data buffer. Every 5 seconds, or when a change of display mode is initiated, the buffer data is written to the current Trending file. By this device, wear on the CMU computer's hard drive (on which the Trending files are kept) is reduced.

7.6.4.2.7. Note that all the functions used to access and control the Trending files are global, and are not contained in a class structure. However, they are all defined in one file - TrendFileControl.cpp, and are all declared in TrendFileControl.h.

7.6.5. Beacon and Polariser Control

7.6.5.1. Beacon receiving systems vary from one antenna to another. Thus the Beacon setup facilities provided must be specific to the antenna to which the CMU is connected. This capability is accomplished by the construction of a 'Beacon and Polariser Setup' form view for each type of installation, contained within an antenna-specific dll.

7.6.5.2. The Beacon and Polariser Setup form view windows are wrapped by the clFrmBeaconccc class objects (where 'ccc' is a character sequence indicating a unique installation), which are derived from clFormView. The window resources and wrapper class objects are provided from within an antenna-specific dll. However, this dll is able to nominate a form view contained within the CMU code. This relates to a relatively common beacon setup, consisting of one receiver and one down converter, both of the MITEC Skylink 2000 series. The class for this form view is clFrmBeaconStyle1. (A matching satellite setup property page, wrapped by the class object clPpSatBeaconPageStyle, is also included in the CMU code.)

7.6.5.1. Function

7.6.5.1.1. Provides access to ATCU settings currently used in the control of the beacon receiving system, and any polariser fitted to the antenna. Allows changing satellite default settings to those currently in use.

7.6.5.2. Dependants

7.6.5.2.1. Because the Beacon and Polariser Control display is antenna-specific, the details of dependant windows will vary. Here, the form view included in CMU is used as a basis, to indicate an appropriate type of arrangement.

7.6.5.2.2. clFrmBeaconStyle1 contains edit and button controls for beacon parameter settings. A covering window wrapped by a CStatic class object is provided, which hides all the beacon controls if no beacon system is connected. A child window contains polariser displays and controls. The resource and wrapper class for this is obtained from a polariser dll nominated by the antenna-specific dll.

7.6.6. System Status

7.6.6.1. Antenna field equipment varies from one antenna to another. Thus the field monitoring facilities provided by the System Status form view must be specific to the antenna to which the CMU is connected. This capability is accomplished by the construction of a 'Antenna System Status' form view for each type of installation, contained within an antenna-specific dll.

7.6.6.1. Function

7.6.6.1.1. Provides an indication of the state of each antenna system monitored by the ATCU (and PLC, if one is fitted). A three-colour control, clColorStatic, is available from within the CMU code, for use in generating each status indicator. This control provides a green red or blank square. Text objects may also be used, for instance to indicate the position of a stowpin.

7.6.6.1.2. The class which wraps the System Status display is generally used as a convenient container for all the decoding of ATCU or PLC data specific to the installation. This task largely consists of determining the state of each alarm or status represented by a bit in special bytes contained in the ATCU and/or PLC messages. Additionally, a required member function initialises character arrays with all the alarm and event messages which are specific to the installation.

7.6.6.2. Dependants

7.6.6.2.1. All the displays in the System Status form view will be children of the form view window.

7.7. The System Configuration Dialog Box

7.7.1. Being generated in response to user action on the application's menu bar, this dialog box is a modal pop-up style box, owned by Main Frame. Thus, when it is visible, no controls outside it can be accessed.

7.7.1. Function

7.7.1.1. Allows selection of one of four serial communications ports for ATCU interaction, and, if a PLC is used, also selection of one of four serial communications ports for PLC interaction. The serial connections may also be broken or re-made.

7.7.1.2. Allows adjustment of the length of time that the ATCU will maintain tracking operations after release of control by the CMU in control.

7.7.2. Dependants

7.7.2.1. All the controls within this dialog box are standard MFC controls, and are children of the box.

7.8. The Set System Time Dialog Box

7.8.1. Being generated in response to user action on the application's menu bar, this dialog box is a pop-up style box, owned by Main Frame. It is generated as a modal box. Thus, when it is visible, no controls outside it can be accessed.

7.8.1. Function

7.8.1.1. Allows the time and date held by the ATCU CMOS clock to be changed. This is possible only if

a. The CMU is in control, and
b. The ATCU is operating on its CMOS clock.

7.8.2. Dependants

7.8.2.1. All the controls within this dialog box are standard MFC controls, and are children of the box.

7.9. The Stowing Functions Dialog Box

7.9.1. This dialog box is brought into being by a click on the Stowing button in the Control Panel. It is non-modal, thus controls outside the box remain accessible whilst the box is visible.

7.9.1. Function

7.9.1.1. Provides control over all the Stowing operations for the antenna. This includes:-

a. Issuing a command to stow;
b. determining whether stowpin action will be initiated automatically when the Stowed condition is reached, and
c. inserting or extracting stowpins.

7.9.1.2. The current position of each stowpin installed is displayed, and whilst a stowpin may be in motion, a progress indicator for it is displayed.

7.9.1.3. As stowpin arrangements may vary between antennae, this dialog box has a flexible appearance. The existence of each stowpin control and indicator is controlled by information received from the ATCU when the CMU receives the initial information about the antenna. Additionally, the timing of any stowpin action progress indicator display is controlled by that information.

7.9.2. Dependants

7.9.2.1. All the controls within this dialog box are standard MFC controls, and are its children.

7.10. Alarm and Event Display

7.10.1. Being generated in response to user action on the application's menu bar, this dialog box is a pop-up style box, owned by Main Frame. It is generated as a non-modal box. Thus, when it is visible, active controls outside it can be accessed.

7.10.1. Function

7.10.1.1. The function of this display is to present a chronological list of current alarms and any pertinent events – for instance a change of ATCU state or mode, or a change of control point. Note that past alarm conditions which have been cleared are not displayed here.

7.10.2. Dependants

7.10.2.1. All controls within the dialog box are standard MFC controls, and are children of the box. The display of alarm and event data is implemented in a List Box.

7.11. The Editor

7.11.1. A separate application – SuperEditor.exe – is provided with CMU.exe. This editor is an enhanced version of Microsoft's SuperPad, which is provided with the Visual C package. Similar in many respects to Microsoft's WordPad (but without the capability of handling Word documents), SuperPad has been augmented to provide a page-heading capability.

7.11.2. SuperEditor is spawned by CMU either in response to a menu command or a sample-viewing requirement generated from the Trending display.

7.11.3. The menu command results in SuperEditor's File Open facility being positioned in the folder holding the event files for the connected antenna (or, if no antenna is connected, in the hard drive's root directory). It is provided as a convenient means of reviewing the event files. However, its use is not restricted in any way.

7.11.4. If the operator makes a selection of a span of the historical Trending chart display, a button for review of the displayed data is presented. Clicking on that button causes SuperEditor to be spawned, and loaded with a file containing the raw data displayed in the selected span. Here, a ready-made page heading is included, and the data is displayed in columns under appropriately named headings.

7.11.1. Function

7.11.1.1. To provide a convenient means of reviewing, copying, editing and/or printing data contained in Event files, or Trending data.

7.11.2. Dependants

7.11.2.1. Although the source code for SuperEditor is available internal detail of its operation is not within the scope of this document.

8. DATA EXCHANGE MESSAGE HANDLING

8.1. The procedures used for message handling are the same, whether the message is a command from the CMU, response from the ATCU (or PLC), or the Regular Poll from the ATCU.

8.2. A request for data results in the following sequence:-

a. Document is asked to send a poll, being handed a code value, which specifies the data involved;

b. Document builds a pre-defined message for the code value provided;

c. Document passes the message to the appropriate serial transmitter thread, via its mailbox, and, for ATCU exchanges, stores the message;

d. The serial thread adds any required protocol (including a checksum, at least for the ATCU communications) to the message before transmitting it;

d. When the associated serial receiver thread collects a message, it checks the message for checksum and protocol conformity, and if these are correct, passes the message to Document, via its mailbox;

e. Document checks that the received message is a response to one it has sent (and stored temporarily), and if so, parses the message and sets the data into the appropriate CMU variables. If all the criteria are not met, and error is notified.

8.3. A command to the ATCU to set data results in a sequence similar to that described above, with the following difference:-

a. On receiving the response, Document compares the received message with the one it sent (and stored temporarily), character-by-character. If the two messages are not identical, an error is notified.

8.4. A command for the ATCU (or PLC) to do something results in a sequence similar to that described for setting data, with the following difference:-

a. Document checks that the received message is a response to a message it has sent (and stored temporarily). If not, an error is notified.

8.1. Satellite-specific Default Settings

8.1.1. The ATCU can store default settings for a number of parameters, for each of the ten satellites provided for. These settings are stored in non-volatile form, in the ATCU. All are accessible through each satellite's Data Dialog Box, which contains a number of Property Pages:-

a. Name and tracking-box limits;

b. OrbitTrack settings;

c. Intelsat settings;

d. NORAD settings;

e. Beacon settings (does not exist if no beacon system is connected), and

f. Polariser settings. (does not exist if the antenna does not have a polariser.)

8.1.2. As a preface to this discussion, a point regarding satellite numbers must be clarified. From common useage, the satellite being used is reflected to the operator as the 'Selected Satellite'.

However, within the CMU coding, it is known as the 'Set Satellite' (the variable inSetSatellite holds its serial number). Again within the CMU coding, the term 'Selected Satellite' (variable inSelectedSatellite) is reserved to indicate the satellite whose default settings are required, or are to be changed. Whereas inSetSatellite may hold a value of 0 to 9 inclusive (representing satellites 1-10), inSelectedSatellite may hold a value of -1 to 9 inclusive. To the ATCU, a satellite number of -1 indicates a reference to the settings currently in use, rather than a satellite's default settings.

8.1.3. None of the satellites' default settings may be changed unless the CMU is in control. If the selected satellite is the currently set satellite, its default settings, other than beacon and polariser settings, cannot be changed if tracking of any type is taking place.

8.1.4. For user convenience, beacon settings and polariser settings currently in use have been made available for change at any time (from the CMU in control), via the CMU's Beacon Setup display. In addition, from that display, the currently set satellite's default settings for these two categories may be changed to those currently in use. This dual-access system requires careful control, and is achieved by the CMU having two structures containing beacon setting data, and two structures containing polariser data. These are, respectively:-

a. stSatelliteBeacon and stCurrentBeaconSettings;
b. stPolariser and stReqdPolariser.

(Note that whereas the two polariser structures are of the same type, the stCurrentBeaconSettings structure's type is a sub-set of the type of stSatelliteBeacon, which contains two structures of the same type as stCurrentBeaconSettings one for each of the two allowed-for beacon receiving sets.)

8.1.5. When current beacon settings are to be changed, the stCurrentBeaconSettings structure is filled with the required values, and sent to the ATCU with a Selected Satelite number of -1. When a satellite's default beacon settings are to be changed, the stSatelliteBeacon structure is filled, and sent to the ATCU with a Selected Satellite number equal to the serial number (0-9) of the satellite whose default settings are to be changed.

8.1.6. When current beacon settings are required to be downloaded from the ATCU, the appropriate message is constructed with a Selected Satellite number of -1, and stCurrentBeaconSettings structure is filled with the values received. When a satellite's default beacon settings are required to be downloaded from the ATCU the appropriate message is constructed with a with a Selected Satellite number equal to the serial number of the satellite whose default settings are required, and stSatelliteBeacon structure is filled with the values received.

8.1.7. When current polariser settings are to be changed, the stReqdPolariser structure is filled with the required values, and sent to the ATCU with a Selected Satelite number of -1. When a satellite's default polariser settings are to be changed, the stReqdPolariser structure is filled, and sent to the ATCU with a Selected Satellite number equal to the serial number of the satellite whose default settings are to be changed.

8.1.8. Current polariser conditions are received in the ATCU's Regular Poll, thus there is no requirement to handle them via special messages. The data obtained from the Regular Poll is used to fill the stPolariser structure.When a satellite's default polariser settings are required to be downloaded from the ATCU, the appropriate message is constructed with a with a Selected

Satelite number equal to the serial number of the satellite whose default settings are required, and the stReqdPolariser structure is filled with the values received.

9. INSTALLATION-SPECIFIC REQUIREMENTS

9.1. Isolation of Unique Code

9.1.1. All code relating to unique aspects of an antenna or polariser installation must be contained in a dll provided for that antenna or polariser.

9.1.2. The central CMU code accesses code within the dlls via a few standard functions, and pointers to class objects coded within the dll. Thus the entire interface with the dlls is standardised, but the detail within each dll may be varied as required.

9.1.3. Access to some global variables and functions within CMU.exe is provided for in the construction of CMU.exe. Qualifiers, defined in the CMU file Definitions.h, provide export and import storage qualifications to these entities. In order that code within a dll can have access to these entities, the library CMU.lib must be included in the link definition for the dll build. This action is set in the Visual C++ workspace by selecting Project/Settings/Link/General, and entering the pathname for CMU.lib in the Object/Library modules window.

9.2. Antenna DLL Requirements

9.2.1. Library Type

9.2.1.1. The name of an antenna dll must be

AntennaTypeN

where N is a digit or series of digits which equals the antenna type coded into the ATCU program for that antenna. The dll name, with the extension .dll added, must be the name of the dll file.

9.2.2. Required Functions

9.2.2.1. An antenna dll must provide the following functions, declared as indicated.

9.2.2.1. Construction Function

```
extern "C" __declspec(dllexport) void fvdConstructUniqueViews(
                clFormView** ppfvBeaconSetup,
                                clFormView** ppfvStatus,
                                        HINSTANCE* phPolariserDLL )
```

This function must construct the Beacon and Polariser Control and the System Status form view wrapper class objects specific to the antenna. Following this action CMU creates the associated windows and attaches them to these objects.
If a polariser is installed in the antenna, it must also attach the required polariser dll.
The function provides, via its parameters, pointers to the class objects for the two form views, and a handle for the polariser dll. If any view is not created, the pointer provided must be NULL. If a polariser dll is not attached, the handle provided must be NULL.
Typically, the code for this function would be:-

```
// Create the formview wrappers.
*ppfvBeaconSetup = new clFrmBeaconStellar;
*ppfvStatus = new clFrmStatusStellar;

// Attach the Polariser Dll and provide its handle.

// Build the name of the required Dynamic Link Library.
CString csPolariserLibName = POLARISER_DLL_NAME;
CString csPolariserLibPath;
csPolariserLibPath = csPolariserLibName + ".Dll";
// Attach the library.
*phPolariserDLL = LoadLibrary( csPolariserLibPath );
```

where:-
clFrmBeaconStellar is defined within the dll, as the wrapper class for a Beacon and Polariser Control form view;
clFrmStatusStellar is defined within the dll, as the wrapper class for a System Status form view, and
POLARISER_DLL_NAME is defined as required, to match the name of the required polariser dll.

This function is called once only, immediately after CMU has attached the antenna library.

9.2.2.2. Dll Version Function

```
extern "C" __declspec(dllexport) void fvdDllVersion( char* pszVersion )
```

This function must supply, via its parameter, a null-terminated character string which reflects the version details of the dll. The function is called whenever the CMU ‘About’ box is to be displayed, and the character string supplied is shown in that ‘About’ box. The string must be no more than 3 lines, each line not exceeding 72 characters.

9.2.2.3. Satellite Data Beacon Page Function

```
extern "C" __declspec(dllexport) CPropertyPage* fpPpUniqueSatBeaconPage()
```

This function must construct the Beacon property page’s wrapper class object for the CMU’s Satellite Beacon Setup dialog box. It must return a pointer to that wrapper class object.
Typically, the code for this function would be:-

```
// Create the satellite data box's beacon page wrapper.
CPropertyPage* pPpPage;
pPpPage = new clPpSatBeaconPageStellar();

return pPpPage;
```

where clPpSatBeaconPageStellar is defined within the dll, as the wrapper class for a Beacon property page.

9.2.2.4. PLC Serial Channel Function

```
extern "C" __declspec(dllexport) CSerial* fclBuildSerialPLCChannel(
                                        BOOL boCreate, UINT uinComPort )
```

NOTE: This function is NOT required to be defined if no PLC is fitted.
This function must create or destroy the serial channel for communication with the PLC, if one is installed. If the first parameter is TRUE, it must create the channel. If the first parameter is FALSE, it must destroy an existing PLC channel. The hardware comport to be used is indicated by the second parameter. Note that these are numbered 1-4, corresponding to COM1-COM4.
The function must return a pointer to the serial channel class object, or NULL if it is not created, or is destroyed.
Typically, the code for this function would be:-

```
CSerial* pclPLCChannel = NULL;
if( boCreate )
        pclPLCChannel = new clPLCSerial( uinComPort,
                                        MAX_PENDING_POLLS,
                                        MAX_MESSAGE_LENGTH,
                                        MAX_PENDING_POLLS,
                                        MAX_MESSAGE_LENGTH, NULL);
else
{
        if( pclPLCChannel != NULL )
        {
                delete pclPLCChannel;
                pclPLCChannel = NULL;
        }
}

return pclPLCChannel;
```

This function is called once after the antenna library is attached, and again when the CMU is terminating, or the ATCU connection has been terminated.

9.2.3. Form View Requirements

9.2.3.1. The two form views provided must be derived from the CMU class clFormView. They will be manipulated through the pointer provided by the construction function, and the following member functions, which are all defined in clFormView as virtual members of that base class.

```
void m_fvdInitialise();
```
Called once immediately after the form view window is created.

```
void m_fvdUpdateDisplay();
```
Called after every Regular Poll is received from the ATCU.

```
void m_fvdLoadUniqueAlarms();
```

This member of the System Status form view class is called once after the view has been built. The same member of the Beacon and Polariser form view class is never called.

BOOL m_fboReadUniqueATCURegularPoll(char* pchDataString,
UINT uinLowestUnusedAlarmWord);
This member of the System Status form view class is just before every call to this view's member m_fvdUpdateDisplay(). The same member of the Beacon and Polariser form view class is never called. A TRUE returned value indicates success.

BOOL m_fboDecodeUniquePLCData(UINT uinPollNumber,
char* pchMessage);
This member of the System Status form view class is called just after every call to its member m_fboReadUniqueATCURegularPoll() and before every call to its member m_fvdUpdateDisplay(). The same member of the Beacon and Polariser form view class is never called. A TRUE returned value indicates success.

void m_fvdUniquePLCMessage(UINT uinPollNumber,
char* pchMessage);
This member of the System Status form view class is called when a requirement to send a message to a PLC is generated. Its first parameter will indicate the required message's code number. It should provide a null-terminated character string via its second parameter.

Note that definition of these functions is not mandatory. Only those appropriate to the installation need be used. It is suggested that existing code be used as a basis for new dll code. The existing source code contains ample comment to indicate the processes involved.

9.2.4. Property Page Requirements

9.2.4.1. The property page provided must be derived from the CMU class clPropertyPage. It will be manipulated through the pointer provided by the construction function, and the following member function, which is defined in clPropertyPage as a virtual member of that base class.

void m_fvdUpdatePageDisplay(BOOL boDisplayUpdateRequired)
This function is called when the page is active, just after each ATCU Regular Poll is received. At this call, the parameter is always FALSE. It is also called when just before the page is 'killed' – for instance after a requirement to change to another page, or to destroy the satellite setup dialog box. At this call, the parameter is always TRUE.
The value of the parameter indicates whether or not an update of the display controls should be carried out regardless of whether updating has been prevented by some other action. Setting it TRUE allows for the correct operation of the 'keep or discard' query system if an attempt is made to kill the page without completing a data change operation.

9.2.5. PLC Serial Channel Requirements

9.2.5.1. A PLC serial channel class shall be derived from the CMU class CSerial. It will be self-contained, and is not manipulated by external calls to its functions. The CSerial class provides all the functionality for serial communications other than the detail of building a message from received characters and evaluating it for protocol errors, or adding protocol to a message to be transmitted. Thus the derived class requires only to add definitions of the two virtual functions defined in Cserial. These are:-

void m_fvdBuildMessage(char* pchChars, unsigned long ulgNumChars);

Called when characters have been received at the connected serial port. The first parameter is a pointer to a buffer containing the characters (not necessarily null-terminated), the second indicates the number of characters in the delivery. This may be from 1 to MAX_MESSAGE_LENGTH – defined in the CMU header file Definitions.h.

The function must detect the start and end of a message, build the entire message and check it for protocol errors. If the message is satisfactory, it must pass it to Document, by calling the CSerial member function

void m_fvdPostMessage(char* pchChars, int inNumChars);

void m_fvdAddProtocol(char* pchMessBuf, int* pinMessLgth);

This function must add any protocol required to the message held in a buffer pointed to by the first parameter. The second parameter indicates the number of characters in that message. The additions are made by adjustment of the contents of the indicated buffer. (It is advisable to take a copy of the buffer contents immediately on entry to the function, adjust the copy and then copy the adjusted message back to the external buffer.)

9.2.6. Required Resources

9.2.6.1. The antenna dll ‘project’ should contain a resources file which provides the definition of the actual window resources required.

9.2.6.2. The form view resources should be dialog resources drawn to the required layout, and able to completely fill the space between the two dialog bars of the main frame. (Note that the view is adjusted automatically to fill that space. The positioning of controls etc. is not affected by that adjustment, so the original drawing must provide the correct location.) All form view resources are drawn in Visual C++’s Resource Editor, with resolution set to 1024x768, and large fonts set. They are drawn as a dialog box, sized 408x183 Dialog Units (DLUs). Property settings should be:-

Font name:	MS sans serif;
Font size:	10;
Position:	0,0;
Style:	Child;
Border:	None;
Styles set:	Context Help, 3-D Look, Control, Transparent, Control parent.

9.2.6.3. The property page resource should be a dialog resource drawn to the required layout, and of the same size as other property pages used in the satellite setup dialog box. All these resources are drawn in Visual C++’s Resource Editor, with resolution set to 1024x768, and large fonts set. They are drawn as a dialog box, sized 260x120 Dialog Units (DLUs). Property settings should be:-

Font name: MS sans serif;
Font size: 8;
Position: 0,0;
Style: Child;
Border: Thin;
Styles set: Title bar, Disabled.

9.3. Polariser Dll Requirements

9.3.1. Library Type

9.3.1.1. An polariser dll must be a MFC *Extended* Dynamic Link Library.

9.3.2. Library Name

9.3.2.1. There is no restriction on the name of a polariser dll. The dll name, with the extension .dll added, must be the name of the dll file.

9.3.3. Required Functions

9.3.3.1. A polariser dll must provide the following functions, declared as indicated.

9.3.3.1. Construction Function

```
extern "C" __declspec(dllexport) clDialog* fclConstructPolariserWindow(
                                BOOL boCreate, CWnd* pParent=0 )
```

This function must either construct or destroy the Polariser wrapper class object specific to the polariser, which is required to appear on the Beacon and Polariser Control form view. The first parameter is TRUE if construction is required.
The function must return a pointer to the class object, which must be derived from the CMU base class clDialog. If the object is not created, or is destroyed, the pointer returned must be NULL. Typically, the code for this function would be:-

```
if( boCreate )
{
        if( pclDlgPolariser == NULL )
                pclDlgPolariser  = new clDlgMyPolariser( pParent );
}
else if( pclDlgPolariser != NULL )
{
        delete pclDlgPolariser;
        pclDlgPolariser = NULL;
}
 return pclDlgPolariser;
```

where:-
pclDlgPolariser is a static pointer to clDialog class, and clDlgMyPolariser is defined within the dll, as the wrapper class for a dialog box.

9.3.3.2. Dll Version Function

```
extern "C" __declspec(dllexport) void fvdDllVersion( char* pszVersion )
```

This function must supply, via its parameter, a null-terminated character string which reflects the version details of the dll. The function is called whenever the CMU 'About' box is to be displayed, and the character string supplied is shown in that 'About' box. The string must be no more than 3 lines, each line not exceeding 72 characters.

9.3.3.3. Satellite Data Polariser Page Function

extern "C" __declspec(dllexport) CPropertyPage*
fpPpUniqueSatPolariserPage()

This function must construct the Polariser property page's wrapper class object for the CMU's Satellite Beacon Setup dialog box. It must return a pointer to that wrapper class object.
Typically, the code for this function would be:-

```
// Create the satellite data box's Polariser page wrapper.
CPropertyPage* pPpPage;
pPpPage = new clPpSatPageMyPolariser;

return pPpPage;
```

where clPpSatPageMyPolariser is defined within the dll, as the wrapper class for a Polariser property page.

9.3.4. Dialog Box Requirements

9.3.4.1. The dialog box provided must be derived from the CMU class clDialog. It will be manipulated through the pointer provided by the construction function, and the following two member functions, which are defined in clDialog as virtual members of that base class.

void m_fvdUpdateDisplay();

Called after every Regular Poll is received from the ATCU, when the Beacon and Polariser form view is being shown.

BOOL m_fboSetNewValue(UINT uinControlID);

This function is called when the Beacon and Polariser Control form view is visible, when the Enter key has been pressed, and the control with the focus is not a child of the Beacon and Polariser Control form view. The parameter provides the ID of the control with the focus, and the function should call the dialog box's OnOk() function if the control indicated is a child of the dialog box. The return value should indicate whether this condition is true.

9.3.5. Property Page Requirements

9.3.5.1. The property page provided must be derived from the CMU class clPropertyPage. It will be manipulated through the pointer provided by the construction function, and the following member function, which is defined in clPropertyPage as a virtual member of that base class.

void m_fvdUpdatePageDisplay(BOOL boDisplayUpdateRequired)

This function is called when the page is active, just after each ATCU Regular Poll is received. At this call, the parameter is always FALSE. It is also called when just before the page is 'killed' – for instance after a requirement to change to another

page, or to destroy the satellite setup dialog box. At this call, the parameter is always TRUE.
The value of the parameter indicates whether or not an update of the display controls should be carried out regardless of whether updating has been prevented by some other action. Setting it TRUE allows for the correct operation of the 'keep or discard' query system if an attempt is made to kill the page without completing a data change operation.

9.3.6. Required Resources

9.3.6.1. A polariser dll 'project' should contain a resources file which provides the definition of the actual window resources required.

9.3.6.2. The dialog box resource should be a dialog resource drawn to the required layout, and of the same size the space provided in the Beacon and Polariser Control form view. This resource should be drawn in Visual C++'s Resource Editor, with resolution set to 1024x768, and large fonts set. It is drawn as a dialog box, sized 140x142 Dialog Units (DLUs). Property settings should be:-

Font name:	MS sans serif;
Font size:	10;
Position:	0,0;
Style:	Child;
Border:	None;
Styles set:	Context Help, Set foreground, 3D-look, Control, Control parent

9.3.6.3. The property page resource should be a dialog resource drawn to the required layout, and of the same size as other property pages used in the satellite setup dialog box. All these resources are drawn in Visual C++'s Resource Editor, with resolution set to 1024x768, and large fonts set. They are drawn as a dialog box, sized 260x120 Dialog Units (DLUs). Property settings should be:-

Font name:	MS sans serif;
Font size:	8;
Position:	0,0;
Style:	Child;
Border:	Thin;
Styles set:	Title bar, Disabled.

9.4. Help System Additions

9.4.1. The only additions to the CMU Help information, required to cover installation-specific antenna or polariser facilities are *context help* notes, and code within the dlls to connect the various windows to the help system. Source code for existing, similar windows will indicate how this is achieved.

9.4.2. The file resource.hm, found in the folder containing each antenna or polariser 'project', must be added to the CMU Help system. This may be done in the MS Help Workshop, with CMU.HLP loaded. Click on the 'Map' button, and 'Add' the new file.

9.4.3. The CMU Help information is written in a file titled CMUHelp.rtf, which exists in the CMU project's Help sub-directory. This may be edited in an rtf-capable editor. However, it has been found that MS WordPad, and MS Word97 are not compatible with the MS Help compiler. MS Word 6 works fine.

9.4.4. CMUHelp.rtf is a file of text with footnotes. The text for a Help item is contained between a footnote reference symbol and a page break. The footnote thus referenced is the control ID name provided for the associated control in resource.hm. To insert an additional Help message, add a footnote reference, using a character such as '#', then write the note. A new vacant line in the list of footnotes is created when this is done. Write in the control ID name (obtained from resource.hm) in the vacant spot. Note that unlike normal footnote useage, the footnote symbol comes *in front* of the associated text. To illustrate:-

The text part of the document might be:-

Drive Control Button
This button controls the Drive system.
<page break>

The corresponding footnote would be:-

#HIDC_DRIVE_BUTTON

where HIDC_DRIVE_BUTTON is the Help resource ID for the button in question (whose resource ID would be IDC_DRIVE_BUTTON). If a context-sensitive Help for the button is demanded, the Help system will then display the text:-

Drive Control Button
This button controls the Drive system.

9.4.5. After CMUHelp.rtf has been changed, the help file must be re-compiled. This may be done in the MS Help workshop, or by a complete re-build of the CMU project.