

DELFT UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering

Telecommunications and Traffic control Systems group

Title: Software for the MIAS experimental Airborne system

Author: R.C. Meijer

Code: A481

Date: February 1993

Abstract: For the MIAS project, software was written to interface an MLS and GPS receiver and an attitude sensor to a computer. Also positioning software was written. The software is described and listed here.

# Contents

Contents .....	-ii-
Abbreviations .....	-iv-
1 Introduction to MIAS software .....	-1-
2 Structure of the MIAS software .....	-2-
3 Operation of MIAS .....	-4-
3.1 Necessary units .....	-4-
3.2 Necessary hardware .....	-4-
3.3 Start up sequence .....	-4-
4 Description of MIAS units .....	-9-
4.1 Turbo Pascal Units .....	-9-
4.2 MIAS units .....	-9-
4.2.1 'MIASGLOB' .....	-10-
4.2.2 'GPSGLOB' .....	-10-
4.2.3 'MLSGLOB' .....	-11-
4.2.4 Program 'miassystem' .....	-11-
4.2.5 'Mias' .....	-12-
4.2.6 'GPS' .....	-17-
4.2.7 'MLS' .....	-20-
4.2.8 'ATT' .....	-22-
4.2.9 'HDG' .....	-24-
4.2.10 'DGPS' .....	-26-
4.2.11 'USER' .....	-29-
4.2.12 'POSCALC' .....	-31-
4.2.13 'GPSCALC' .....	-33-
4.2.14 'MATHX' .....	-37-
4.2.15 'MATRIX' .....	-39-
4.2.16 'GPSENGINE' .....	-40-

4.2.17 'MLSBENDIX' .....	-48-
4.2.18 'ATTBEAVER' .....	-57-
4.2.19 'HDGBEAVER' .....	-59-
4.2.20 'KEY_CONS' .....	-61-
4.2.21 'SYNCHCNV' .....	-64-
4.2.22 'AR429' .....	-66-
4.2.23 'AR429COMM' .....	-68-
4.2.24 'ADW' .....	-70-
4.2.25 'MISCELL' .....	-74-
4.2.26 'COM_4' .....	-77-
 Index .....	 -82-
 Appendix A Configuration file format .....	 -83-
 Appendix B Log file format .....	 -84-
 Appendix C Helpful programs .....	 -86-
 Appendix D Listings for MIASLOGO .....	 -87-
 Appendix E Listings for MIASNAV .....	 -88-

## Abbreviations

ADW	Auxiliary Data Words
BDW	Basic Data Word
CG	Centre of Gravity
CR	Carriage Return
DGPS	Differential GPS
DH	Decision Height
DME/P	Distance Measuring Equipment, Precision
DPSK	Differential Phase Shift Keying
EIA	Electronic Industries Association
FCC	Flight Control Computer
FSK	Frequency Shift Keying
GPS	Global Positioning System
HOW	Hand-Over Word
HDG	Heading
ICAO	International Civil Aviation Organisation
ILS	Instrument Landing System
IRS	Inertial Reference System
LF	Line Feed
MIAS	MLS Integrated Approach System
MLS	Microwave Landing System
MSL	Mean Sea Level
OSI	Open Systems Interconnection
PPS	Precise Positioning Service
PRN	Pseudo Random Noise
RVR	Runway Visual Range
SA	Selective Availability
SPS	Standard Positioning Service
TOW	Time of Week
UART	Universal Asynchronous Receiver Transmitter
WGS-84	World Geodetic System 1984
X1 epoch	All ones epoch



# **1 Introduction to MIAS software**

For the MIAS project it was necessary to interface a MLS, a GPS receiver and an attitude sensor to a computer. The information from the mentioned devices is used to calculate the position of the aircraft where the devices are mounted.

In this document, the software which was written to interface with the devices and to calculate the position of the aircraft, is treated. First a close look is taken to the structure of the software and the possibilities to expand or change the software. Then each unit of the MIAS software is described. All procedures and functions in the units are described also. Finally an index is given for cross-reference with all procedure and function names with references to the unit they are in. The appendices contain the listings of the MIAS software plus a description of the log file format and a description of the configuration file format. One appendix lists some helpful programs for MIAS.

Two versions exist for the MIAS-program: a logging only version and a positioning and logging version. The first is a stripped version of the second and is therefore much faster and can be used on a slower computer. This version only logs information to disc and is called 'MIASLOGO'. The second version logs information to disc, but also calculates the position and outputs it to disc. This version is called 'MIASSYST'.

## 2 Structure of the MIAS software

The software for the MIAS airborne part is designed to be highly modular and hierarchical. This means that a lot of effort was put in making the separate pieces of software testable on itself. The software was designed so that it could easily be expanded and parts can be replaced by better parts. Also the use of other hardware is easy by replacing the hardware specific parts and inserting a new version.

Figure 2.1 shows the connections between the main blocks of the MIAS airborne software. A block can use functions or procedures from a block below, which is connected with a line. This allows a top-down approach of the design problem. Each problem is divided into several sub problems, which are solved in a lower block in the figure. This division is continued until basic statements can be used to solve the sub problem. An example of this is given in figure 2.2.

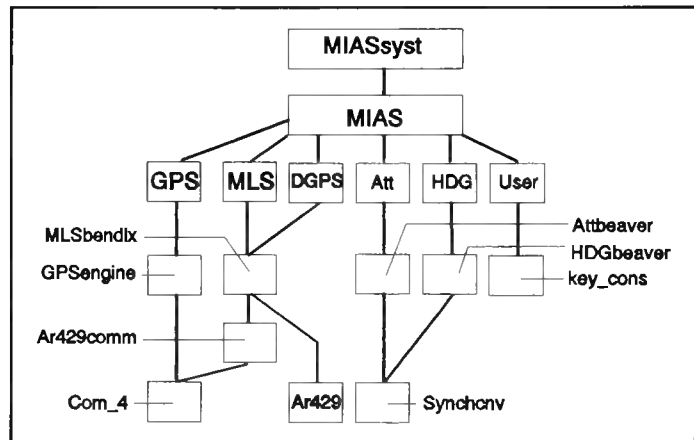


Figure 2.1 The connections between the main block of the MIAS software.

init	- init MLS	->->->->	init MLS receiver	->->->->	init ARINC 429 card
	- init GPS	-	-> init variables		
	- init DGPS	-			
	- init Att	-			
	- init Hdg	-			

Figure 2.2 Example of problems and sub problems.

Because a large part of the software consists of interfacing and communication between the computer and its input devices, this way of solving problems can be seen as a quasi OSI<sup>1</sup>-model. The information comes in from the bottom of the hierarchy and is converted to digital words and variables as it bubbles up in the hierarchy.

An example of the OSI approach is given figure 2.3, where level zero is the electrical level, level one is the driver level, where the bits and bytes are collected from the interfaces. Level two is an adjustment level. In level three, the bits and bytes are grouped in messages, which are converted to variables. In level four, the calculations for the subsystems are performed and finally in level five, the position is calculated.

<sup>1</sup> OSI is Open Systems Interconnection

It must be noted that in figure 2.1 some blocks are not shown. These blocks have no functionality for core MIAS activities. They contain only types and constants, routines for displaying and storing information, and timer and configuration file related routines. Showing them would make the figure confusing. The units not shown are: MIASGLOB, MLSGLOB, GPSGLOB, USER, KEY\_CONS and MISCELL.

The interface units are all build using the same set of procedures. All interface units have an 'init' procedure, a 'getdata' procedure, a 'execcommand' procedure and a 'close' procedure.

The 'init' procedure initialises its peripheral, so it can be used afterwards. This means that hardware interfaces are made active and initialising parameters are sent to the peripheral.

'Getdata' procedures collect data, that was sent by the peripheral to the hardware interface. These data are normally bytes or digital words. This digital information is then converted to variables, which can be used further in the program.

'Execcommand' procedures try to send or execute the command with which they are called. Usually it means that a message should be send to the peripheral, saying it should do or change something.

A 'close' procedure puts the peripheral and hardware interface to their original state.

If more units are added, these units should contain at least these procedures. It could be wise to implement only these procedures, because these four procedure are enough to initialise the device, to communicate with it and to close it again.

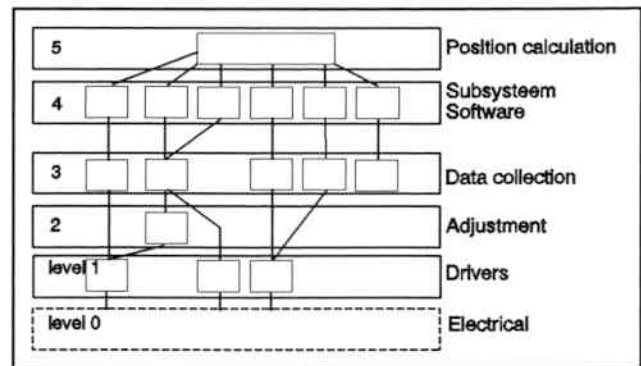


Figure 2.3 Example of an OSI approach of MIAS.

## 3 Operation of MIAS

In this section, the necessary software units are listed as well as the necessary pieces of hardware, needed to run the two versions of the MIAS program. Also a start up sequence is given and the operator interface is described. The software was written in Turbo Pascal 6.0.

### *3.1 Necessary units*

Table 3.1 lists the filenames of the program and units needed to run the MIAS program. The easiest way of using them is to copy them to a special directory. Then the Turbo Pascal IDE<sup>1</sup> should be executed from that directory. By specifying the unit and include directories<sup>2</sup> and selecting 'build'<sup>3</sup> and 'run'<sup>4</sup>, the program will be build and run.

If the executable of the MIAS program is available, it can be run from any directory. Be sure the file 'mias.cfg' is in the same directory as the executable is. If it is not, the program will not run. The 'MIASLOGO'-program uses files with the same names as the 'MIASSYST'-program, but some of these files are stripped. Files for 'MIASLOGO' and 'MIASSYST' are in separate directories.

### *3.2 Necessary hardware*

The hardware listed in table 3.2 is the hardware used for the MIAS concept until now. Of course other hardware can be used, but it requires adjustments to the software.

### *3.3 Start up sequence*

The start up sequence for the MIAS program and its hardware is not very critical, but following the steps as they are shown in list 3.3, ensures that everything works properly. List 3.4 shows the contents of the

---

<sup>1</sup> Integrated Development Environment (IDE). The Turbo Pascal IDE is executed by typing 'turbo' at the DOS prompt.

<sup>2</sup> Select 'Options' at the top menu bars. Select 'Directories' from the pull down menu.

<sup>3</sup> Select 'Compile' from the menu bar. Select 'Build' from the pull down menu.

<sup>4</sup> Select 'Run' from the menu bar. Select 'Run' from the pull down menu. Or press CTRL-F9.

Table 3.1 The necessary file to run MIAS

Filename	Unit/Program name	Function	Needed for MIASSYST	Needed for MIASLOGO
Miassyst.pas	Miassystem	Program	y	n
Miaslogo.pas	Miaslogonly	Program	n	y
Mias.pas	Mias	Unit	y	y
Gps.pas	Gps	Unit	y	y
Mls.pas	Mls	Unit	y	y
Dgps.pas	Dgps	Unit	y	n
Att.pas	Att	Unit	y	y
Hdg.pas	Hdg	Unit	y	y
User.pas	User	Unit	y	y
Gpsengin.pas	Gpsengine	Unit	y	y
Mlsbendi.pas	Mlsbendix	Unit	y	y
Attbeave.pas	Attbeaver	Unit	y	y
Hdgbeave.pas	Hdgbeaver	Unit	y	y
Key_cons.pas	Key_cons	Unit	y	y
Com_4.pas	Com_4	Unit	y	y
Ar429.pas	Ar429	Unit	y	y
Ar429com.pas	Ar429comm	Unit	y	y
Synchcnv.pas	Synchcnv	Unit	y	y
Poscalc.pas	Poscalc	Unit	y	n
Gpscalc.pas	Gpscalc	Unit	y	n
Matrix.pas	Matrix	Unit	y	n
Matrix.inc	-	Include file	y	n
Mathx.pas	Mathx	Unit	y	n
Miasglob.pas	Miasglob	Unit	y	y
Gpsglob.pas	Gpsglob	Unit	y	y
Mlsglob.pas	Mlsglob	Unit	y	y
Mias.cfg	-	Configuration file	y	y
Adw.pas	Adw	Unit	y	y
Intrupt.pas	Intrupt	Unit	y	y

configuration file 'MIAS.CFG', for using the Magnavox GPS Engine receiver, the Bendix MLS 20-A receiver and the synchro-to-digital converter.

### 3.4 Display contents

When the 'MIAS' program is running, it will display all the information it receives. Each input-device has its own one or two lines on the display which show the data currently being received.

Besides the input data, also the position fixes are shown on the screen. Furthermore, there is a line which indicates the status of the input devices. An message saying 'err' indicates that the input does not output data or is not connected. The indication 'oke' tells the user that the input device outputs data.

Table 3.2 Necessary hardware

Function	Type, serial number etc
GPS receiver	Magnavox MX4200D partnr. 900555-803 sernr. 782
MLS receiver	Bendix MLS 20-A
Synchro to digital converter	Using a RDC 19220
Computer	IBM-PC compatible 80486, 33 Mhz DX with 4 MB
Display	Philips LCD with associated interface card
ARINC 429 interface	Max Technologie M429PC/1
DPSK interface	B/ADW serial to parallel converter with interrupt on a preprogrammed bit pattern
RS 422 interface	Comport compatible interface. Interrupt on IRQ 4 and IRQ 3

Figure 3.1 gives an example of the output of the screen when running.

```

312041.00

Number of GPS equations: 2 Number of MLS equations: 0
rollangle = 359.99 pitchangle = 359.99 hdgangle = 359.99

$PMVXG,001,144033,5159.942,N,00422.403,E,00087.9,3*40
 2 J>T/KPéhijfC-=9wd0ÄaT[z.Y,az=WñÆX.(Z~NâI=áæi;>}ó=Z)CêYaæu:+-â4âÉmëO8r'LZfEr
GPS err DGPS err MLS err Att oke HDG oke Pos err int err
Time set to User time of GPS receiver

```

Figure 3.1 Example of the screen contents when MIAS is running.

Table 3.3 Start up sequence for the MIAS logging program.

- 
0. Start plane engine.
  1. Switch on main power
  2. Switch converter power on
  3. Switch PC power on
  4. Insert floppy with programs
  5. Change directory to 'A:\MIASEXE'
  6. Run 'MIASLOGO' or 'LOG'
  7. Remove the floppy
  8. Insert new floppy containing 'MIAS.CFG'
  9. Press enter  
MIAS LOG-FLOPPY LOADED
  10. Switch MX4200D power on
  11. Check if the MX4200D outputs data on both ports.
  12. If not: type 'STOP', press Enter when asked, remove floppy, switch PC power off and goto 3  
GPS WORKING
  13. Turn MLS-20A mode-knob to test
  14. Check if 'PASS' appears on the MLS-CDU within 8 seconds
  15. If not: turn MLS-20A mode-knob to off, and goto 14  
MLS WORKING
  16. Turn MLS-20A mode-knob to 'AUTO'
  17. Select MLS channel using the 'Select'-knob. Channel is:548  
MLS CHANNEL SELECTED
  18. Check if the attitude and heading angles are stable: at least 1 degree.
  19. If not: angles not available.  
ATTITUDE AND HEADING WORKING
  20. Check the computer screen on disc messages
  21. If disc full: remove logging disc and insert new disc.
  22. To stop: type 'STOP'
  23. Remove floppy disc.
  24. Turn MLS-20A mode-knob to 'OFF'
  25. Turn MX4200D power off
  26. Turn PC power off
  27. Turn Converter power off.
-

Table 3.4 Example of MIAS configuration file.

---

**MIAS**

```

allowed_error = 1.0000000000000E-0003;
position.wgs84lat = 9.07562345003371E-0001;
position.wgs84lon = 7.63406910948859E-0002;
position.wgs84alt = 1.97506979882231E+0002;
dgpsmode = 3;
alldata.mls.mlsthrespos.wgs84lat = 0.000000000E+0000;
alldata.mls.mlsthrespos.wgs84lon = 0.000000000E+0000;
alldata.mls.mlsthrespos.wgs84alt = 0.000000000E+0000;
alldata.pos_zerovector.x = 0.0000000000000E+0000;
alldata.pos_zerovector.y = 0.0000000000000E+0000;
alldata.pos_zerovector.z = 0.0000000000000E+0000;
alldata.ant_zerovector.x = 0.0000000000000E+0000;
alldata.ant_zerovector.y = 0.0000000000000E+0000;
alldata.ant_zerovector.z = 0.0000000000000E+0000;
alldata.gps.present = 1;
alldata.dgps.present = 0;
alldata.mls.present = 1;
alldata.att.present = 1;
alldata.hdg.present = 1;
drive = a;;

```

**GPS**

```

position.wgs84lat = 9.07562345003371E-0001;
position.wgs84lon = 7.63406910948859E-0002;
position.wgs84alt = 1.97506979882231E+0002;
el_limit = 5;
horaccfac = 1.0000000000000E+0001;

```

**GPSENGINE**

```

port0 = 2;
port1 = 1;
completeinfo = 1;

```

**AR429COMM**

```

mlsport = 3;

```

**MLSBENDIX**

```

completeinfo = 1;

```

---



## 4 Description of MIAS units

In this section the MIAS software units are described. First a brief introduction to the Turbo Pascal feature called "unit" is given.

### *4.1 Turbo Pascal Units*

The software written for the MIAS airborne part is written in Turbo Pascal 6.0. This language allows the division of a program into "unit's". Each unit has an interface section and an implementation section. The interface section defines the types, constants, variable, functions and procedure calls that can be accessed from programs that use the "unit". The implementation section contains the code for these functions and procedures and possibly extra functions and procedures that cannot be accessed by other programs. There is also an initialisation section, which is run only when starting of the program.

### *4.2 MIAS units*

In this section the MIAS program and the MIAS units are described in a top-down way.

The sections which describe a unit, will start with the unit name. Then the unit is described shortly and the in and outputs of the units are listed as well as the names of the units that are used and the names of the units that use this unit. One file contains one unit. The file name is an abbreviation of the unit name.

Every following section, which describes a function or a procedure from the unit, will start with the definition of the function or procedure call. Then a short description of the action inside that routine is given followed by a short description of the in and outputs of the routine. Then the routines are listed which call the routine which is being described.

### 4.2.1 'MIASGLOB'

This unit contains constants and variable types to be used by many other unit in the MIAS program.

Input	:	-
Output :	-	
Used by	:	'miassystem', 'mias', 'gps', 'dgps', 'mls', 'att', 'hdg', 'poscalc', 'gpsengine', 'mlsbendix', 'attbeaver', 'hdgbeaver', 'user', 'key_cons' and 'synchcnv'.
Uses	:	'miscell'

Note: The 'double' type is the Turbo Pascal 'Extended' type.

### 4.2.2 'GPSGLOB'

This unit contains constants and type declarations for the GPS part of the MIAS system. The constants with the name starting with 'Valid\_T' are used to supervise the GPS information. These constants contain the duration for which the information is valid.

The 'ephemeris' type is a record containing a field for every ephemeris parameter. The same goes for the clock parameters. Per satellite, there is a record containing pseudorange, health, ephemeris, clock, transmission and reception time, integrated carrierphase, elevation and azimuth of the user to the satellite, eccentric anomaly of the satellite orbit, a flag, the satellite position and the time of reception of clock and ephemeris parameters. A record is also reserved for ionosphere parameters.

The 'GPSint' type, is a record which contains a flag, an array for 32 satellites, the number of satellites being tracked, ionosphere parameters and the ionosphere parameter reception time.

Input	:	-
Output :	-	
Used by	:	'gps', 'gpsengine', 'gpscalc'
Uses	:	'miasglob' and 'miscell'

### 4.2.3 'MLSGLOB'

This unit contains constants and types for the MLS part of the MIAS program. The constant which name begin with 'Valid\_T' contain the duration that the parameter is valid. Then some constants are listed, which are used to replace the basic data words, which might not be available when using some types of MLS receivers.

Furthermore, for every defined basic and auxiliary data word there is a record defined, containing the corresponding data fields. The discretes type was defined to allow an ARINC 727 MLS receiver to be used.

The 'mlsint' type contains all basic and auxiliary records, with associated reception time and flags. It also contains the measured angles and DME range, discretes, left and right clearances, antenna selection and an overall flag.

Note: More information about the data fields can be found in [1, sect 3.11].

Input	:	-
Output :	-	
Used by	:	'mls' and 'mlsbendix'
Uses	:	'miasglob' and 'miscell'

### 4.2.4 Program 'miassystem'

The section 'miassystem' is the actual program. It declares some variables and begins with initialising the MIAS system. 'miassystem' uses compiler directives to increase the standard stack size to 32000 bytes. The maximum heap size is the original maximum heap size.

After initialising 'miassystem' runs a loop, which is terminated by typing the word 'stop'. After terminating the loop, the MIAS system is closed down. During closing down, 'miassystem' parameters are written in the file 'mias.cfg'. 'Mias.cfg' is described in appendix A.

Input	:	-
Output :	-	
Used by	:	-
Uses	:	'miasglob' and 'mias'

### 4.2.5 'Mias'

This unit contains most of the intelligence specific for the 'miassystem'. It translates the calls from 'miassystem', which are very general to more specific calls. For example: the call 'getdata' will be translated to: 'getdgpsdata', etc. It forms the connection to all different subsystems. These are: GPS, MLS, DGPS, ATT, HDG.

'Mias' uses compiler directives to install coprocessor support ({ $\$N+$ }). In case the coprocessor is not present, it will be emulated (,E+}).

The section 'mias' uses many units; for every subsection one, plus one for the hybrid position calculation called: 'poscalc', one for miscellaneous functions called: 'miscell' and the unit called 'crt', which is provided with Turbo Pascal compiler for reading the keyboard and writing to the screen.

Input	:	-
Output	:	-
Used by	:	'miassystem'
Uses	:	'miasglob', 'gps', 'dgps', 'mls', 'att', 'hdg', 'user', 'poscalc', 'miscell' and 'crt'

The functions and procedures in 'mias' will be described now.

#### 4.2.5.1 'Init( var alldata: alldatatype);'

'Init' starts with assigning initial values to variables, which are global to the 'mias' unit. The procedure 'init' in 'mias' checks if the configuration file 'mias.cfg' is present. If not, the system will halt. If 'mias.cfg' is present, then the file is scanned for the word 'mias'. As soon as the word is found, the following lines will be read. See also 'miscell' and appendix A. The line which is read, is interpreted and the variable, which name is mentioned in the line will get a value which is also mentioned in the line.

This way a number of global variables will be assigned:

allowed_error	{ position error allowed in iteration}
position	{ contains position coordinates}
dgpsmode	{ 1: no dgps; 2:no change; 3: always dgps}
alldata.mls.mlsthrespos	{ the position of the origin of the MLS reference system}
alldata.pos_zerovector	{ vector from the MLS antenna to the reference point on the aircraft}
alldata.ant_zerovector	{ vector from the MLS antenna to the GPS antenna}

alldata.gps.present	{ 1: GPS receiver present; 0: not}
alldata.dgps.present	{ 1: GPS receiver present; 0: not}
alldata.mls.present	{ 1: GPS receiver present; 0: not}
alldata.att.present	{ 1: GPS receiver present; 0: not}
alldata.hdg.present	{ 1: GPS receiver present; 0: not}

Then 'init' will call the initialising procedures of all the subsystems. After initialising the subsystems, 'init' will display the flags as determined by the subsystems.

'init' is called by 'miassystem'.

#### **4.2.5.2 'Dispflags( alldata: alldatatype);'**

The procedure 'dispflags' will show the flags of MIAS on an output device. A flag in MIAS is like a flag on a conventional aircraft instrument. As soon as it is raised (True), an error occurred.

The procedure compiles an ASCII message containing all the flag messages for all subsystems one. The line, which results, is passed to the procedure 'sendusermessage'.

'Dispflags' is called by 'init' and 'miassystem'.

#### **4.2.5.3 'Getusercommands( var command: commandtype);'**

This procedure will collect the user inputs from the input device. This procedure is almost empty, it only contains one call to the procedure 'getusermessage'. Though this procedure would not have been necessary, it is included to support the hierarchical way the MIAS system was designed.

'Getusercommands' is called by 'miassystem'

#### **4.2.5.4 'Execcommands( command: commandtype);'**

This procedure will receive a command string, which will start with a header specifying for which subsystem the command is meant. For example: 'MIAS:DGPSMODE = 3;' or 'GPS:RESET;'. Many commands are possible this way, but not many commands are implemented yet. In the unit 'mias' only the

command 'MIAS:DGPSMODE = x;' is implemented. It will assign the value x to the variable 'dgpsmode', which meaning is explained in section 4.2.5.1.

The procedure will determine for which subsystem the command is meant, delete the header and pass the rest of the command to the 'execSUBSYSTEMcommand'-procedure, which is the procedure for executing commands for a specific subsystem.

'Execcommands' is called by 'miassystem'

#### **4.2.5.5 'Getdata( var alldata: alldatatype);'**

'Getdata' collects the sensor information from all sensors available. That is: GPS, DGPS, MLS, ATT and HDG.

Note: At this moment, the attitude and heading is only collected as soon as the GPS flag is false. This is done to prevent the output file to get swamped with attitude and heading info, because it would be collected many times.

'Getdata' is called by 'miassystem'

#### **4.2.5.6 'Calcpos( alldata: alldatatype; var position: positiontype);'**

'Calcpos' calculates the position of a point on the aircraft. To do this, it calls 'calcmls', then it calls 'calcgps' which calculates the satellite positions etc. after which it calls 'calcdgps' which calculates the differential GPS corrections. Finally it calls 'calchybridpos', which determines the position with a least squares algorithm.

Note: Old MLS information is used, to calculate a position. This should be changed. A estimating algorithm should be implemented, so more accurate MLS, GPS and attitude info can be used for positioning.

Note: Some variables and constants are not needed any more.

'Calcpos' is called by 'miassystem'

#### **4.2.5.7 'Filterposition( position: positiontype; var filtposition: positiontype);'**

This procedure is meant to contain a position filtering algorithm, to filter the resulting position. It is empty for now.

'Filterposition' is called by 'miassystem'.

#### **4.2.5.8 'Predictposition( position: positiontype; var predposition: positiontype);'**

This procedure is meant to contain a prediction algorithm, to account for acquisition and calculation delays. It is empty for now.

'Predictposition' is called by 'miassystem'.

#### **4.2.5.9 'Sendposition( position: positiontype);'**

This procedure packs the WGS-84 coordinates of the position in a character-string and sends it to the output device. It only packs and sends the coordinates if the position is valid. If the field 'EcefTrueLocalFalse' is 'true' the position is packed in latitude, longitude, altitude as well as the ECEF coordinates x,y and z. If the field is 'false', only the local coordinates x,y and z are packed.

Note: 'a' is used for local coordinates instead of 'x', to distinguish the different reference systems.

'Sendposition' is called by 'miassystem'

#### **4.2.5.10 'Stopcommand( command: commandtype): boolean;'**

This function returns a boolean. If the command string which is received is 'STOP', the boolean is set to 'true' otherwise it is set to 'false'. This command is meant to be used by the user via the input device. This function is used in the main program loop. As soon as 'stopcommand' is 'true' the program is stopped.

'Stopcommand' is called by 'miassystem'

#### **4.2.5.11 'Closedown( alldata: alldatatype; position: positiontype);'**

This procedure opens the configuration file and writes the variables that are global to 'mias' in the file. Then the sensors that are present are closed one by one.

'Closedown' is called by 'miassystem'.

#### **4.2.5.12 'SetTimetogpsifnotset;'**

This procedure will change the computer system time so it indicates the GPS time as it is indicated by the user time in the GPS receiver. If the time was set to GPS time, indicated by 'timeset', nothing is done. 'Getgpstime' retrieves the GPS time from the GPS receiver.

The GPS time is given as milliseconds into the week. From this time, the hour, the minute etc are calculated and programmed in the computer. Finally a message is send to the screen to tell that the computer clock is now synchronised with the GPS receiver clock.

'Settimetogpsifnotset' is called by 'miassystem'.



## 4.2.6 'GPS'

The unit 'gps' contains global routines for collecting and processing GPS data. The unit 'gps' is part of the backbone of the miassystem. 'Gps' uses a specialised unit for the equipment specific tasks. Furthermore, it uses a unit called 'gpsglob', which contains the GPS types and 'gpscalc', which contains the specific GPS calculation routines for satellite position etc.

'Gps' uses compiler directives, to install coprocessor support ({ $\$N+$ }). In case the coprocessor is not present, it will be emulated (,E+}).

'Gps' declares 4 global variables for the 'gps' unit:

GPSint:	contains all GPS information available
x:	a counter for initialisation
El_limit:	contains the minimum elevation angle allowed for GPS satellites
HorrAccFac:	contains the maximum horizontal acceleration factor.

The last two variables are used to program the GPS receiver.

Input	:	-
Output	:	-
Used by	:	'mias'
Uses	:	'miasglob', 'crt', 'gpsengine', 'gpscalc', 'gpsglob', 'miscell'

### 4.2.6.1 'Initgps( var gpsdata: gpsdatatype);'

This procedure clears all necessary GPS variables. Then the procedure checks the variable 'gpsdata.present'. If this variable equals 1, the GPS receiver is present. If it equals 0, no GPS receiver is connected. Then the configuration file is opened and the program will search for the word 'GPS'. If the word is found, the following variables will be assigned:

position	{ Initial position for GPS receiver}
el_limit	{ minimum elevation angle for GPS satellites}
horaccfac	{ horizontal acceleration factor of GPS receiver}

Then the GPS receiver is commanded to reset and initialising commands are sent to the GPS receiver. Also a request for Ephemeris and Almanac is sent.

'Initgps' is called by 'init'

#### **4.2.6.2 'Getgpsdata( var gpsdata: gpsdatatype; dgps: boolean);'**

This procedure collects data from the equipment specific unit for the GPS receiver. As soon as information is received, the information is checked to see if a time out has occurred. Then the procedure checks if a pseudorange has been received (this has set the 'prn[x].flag' to 'false'). Then it checks if valid satellite clock corrections are received. If so, the ephemeris and health information is checked. If at least one satellite has complete information, the 'gpsdata.flag' is set to 'false', to indicate correct information.

If the 'dgps' flag is 'false', extra information is needed. In that case the ionospheric corrections should be received also.

'Getgpsdata' is called by 'getdata'

#### **4.2.6.3 'Calcgps( var gpsdata: gpsdatatype; position: positiontype; dgps: Boolean);'**

This procedure calls the necessary GPS calculations. If 'gps.flag' is 'true' indicating 'no valid info', the procedure is exit. For every valid satellite, the satellite clock is corrected for its errors and the group delay and the satellite positions are calculated. The pseudorange is calculated also.

If a position fix is available and 'dgps' is 'false', the pseudorange and the transmission time are corrected for ionospheric and tropospheric effects.

Note: Tropospheric effects are not corrected, because the height above Mean Sea Level (MSL) is needed. Height above MSL is different from the WGS-84 height.

'Calcgps' is called by 'calcpos'

#### **4.2.6.4 'Execgpscommand( command: commandtype);'**

This procedure only passes the command to the GPS receiver specific software. So it exists only for the hierarchical system setup.

#### **4.2.6.5 'Closegps( gpsdata: gpsdatatype; position: positiontype);'**

This procedure starts with saving the global gps variables (see 'initgps'). Then it ends with calling the closing routine for the GPS receiver.

'Closegps' is called by 'closedown'.

#### **4.2.6.6 'Getgpstime( var gpstime: Longint; var valid: boolean);'**

This procedure will assign the GPS time of the GPS receiver to 'gpstime'. If no valid pseudo ranges are available, 'valid' will be 'false' and the procedure is exit.

If there are valid pseudo ranges, all pseudo range flags are searched to find valid pseudo range. If one is found the GPS receiver time is copied to 'gpstime' and 'valid' becomes 'true'.

'Getgpstime' is called by 'settimetogpsifnotset'.

## 4.2.7 'MLS'

The 'MLS' unit provides the general MLS functions. For more receiver specific functions, a unit called 'Mlsbendix' is used. 'Mls' uses types which are defined in 'mlsglob'.

'MLS' uses a compiler directive, which causes the mathematic coprocessor to be used if it is present: ({ $\$N+$ }), or it starts emulating the coprocessor if it is not available: (,E+}).

'MLS' declares one variable for global use in the MLS portion of the MIAS program: 'mlsint'. It contains all information from the MLS receiver, like azimuth- and elevation angles, basic and auxiliary data words and flags. In the initialising part of the unit, (almost) all fields of 'mlsint' are cleared.

Input	:	-
Output :	-	
Used by	:	'mias'
Uses	:	'miasglob', 'mlsbendix', 'mlsglob', 'miscell', 'crt'

### 4.2.7.1 'Initmls( var mlsdata: mlsdatatype);'

This procedure clears the flag in the 'mlsint' variable. It then checks if the MLS receiver is really available by checking the variable 'mlsdata.present'. If this variable equals 1, the MLS receiver is available, if it equals 0, the MLS receiver is not available. Then it opens the configuration file 'mias.cfg', where it tries to find the word 'MLS'. Because no MLS variable can be programmed from the configuration file at this time, this part of the program is added to assure easy expansion. Finally the receiver dependent software is signalled to initialise the MLS receiver.

'Initmls' is called by 'init'

### 4.2.7.2 'Getmlsdata( var mlsdata: mlsdatatype);'

This procedure starts the collection of MLS data. It calls the procedure 'collectmlsrec', which gets data from the MLS receiver and puts it in the variable 'mlsint'. Then the data is checked for time outs. That is: every field in 'mlsint' is valid for only a certain period of time. If the field is not refreshed before that time is over, that field will become invalid and the flag concerning that field will be set to 'true'.

Because MLS does not require all data fields to be valid, the overall flag will be set using the allowed subsets of flags. Only one valid function will cause the overall flag to be set 'false'. Then the information is copied from the internal variable 'mlsint' to the global variable 'alldata.mls'.

'Getmlsdata' is called by 'getdata'.

#### **4.2.7.3 'Calcmls( var mlsdata: mlsdatatype);'**

In this procedure, the position of the azimuth, elevation, back azimuth and DME transmitters is calculated in the local reference system. This is done by taking specific fields from the Basic and Auxiliary data words and combining them. Some flags are checked to see if the information is valid.

Then the angle and distance information from MLS is checked. If the signals are available (according to the ground stations) the information is copied to the global variable 'alldata.mls'. Then the master flag 'alldata.mls.flag' is set to indicate if valid information is present. Then the runway heading is calculated.

Note: Until now, the runway heading is the magnetic heading. In fact the true heading is needed.

'Calcmls' is called by 'calcpas'

#### **4.2.7.4 'Execmlscommand( command: commandtype);'**

This procedure is here only to keep up the hierarchy. It only passes the command to the procedure 'execmlsrecommand'.

'Execmlscommand' is called by 'execcommands'.

#### **4.2.7.5 'Closemls( mlsdata: mlsdatatype);'**

This procedure is here only to keep up the hierarchy. It only calls the 'closemlsrec' procedure. Because it should be possible to store some information from MLS on disc for later use, the variable 'mlsdata' is passed. See also 'initmls'.

'Closemls' is called by 'closedown'.

## 4.2.8 'ATT'

This unit provides simple interfacing with an attitude (pitch and roll) indicator. By using the compiler directives '{N+,E+}', the compiler is instructed to compile for mathematic coprocessor if present. If it is not present, an emulating library is included.

Input	:	-
Output :	-	
Used by	:	'mias'
Uses	:	'miasglob', 'attbeaver'

### 4.2.8.1 'Initatt( var attdata: attdatatype);'

Several attitude parameters are cleared in this procedure. Then the 'att.present' variable is checked. If it is 1, the attitude is available. If it is 0, it is not available. Finally, the attitude specific software is instructed to initialise the attitude sensor.

'Initatt' is called by 'init'.

### 4.2.8.2 'Getattdata( var attdata: attdatatype);'

This procedure is here to keep up the hierarchy. It only calls the procedure 'collectatt'.

'Getattdata' is called by 'getdata'.

### 4.2.8.3 'Execattcommand( command: commandtype);'

This procedure is here to keep up the hierarchy. It only calls the procedure 'execatttxcommand'.

'Execattcommand' is called by 'execommands'.

#### **4.2.8.4 'Closeatt( attdata: attdatatype);'**

This procedure is here to keep up the hierarchy. It only calls the procedure 'closeatttx'.

'Closeatt' is called by 'closedown'.

## 4.2.9 'HDG'

This unit provides simple interfacing with an heading indicator. By using the compiler directives '{N+,E+}', the compiler is instructed to compile for mathematic coprocessor if present. If it is not present, an emulating library is be included.

Input	:	-
Output :	-	
Used by	:	'mias'
Uses	:	'miasglob', 'attbeaver'

### 4.2.9.1 'Inithdg( var hdgdata: hdgdatatype);'

Several heading parameters are cleared in this procedure. Then the 'hdg.present' variable is checked. If it is 1, the heading is available. If it is 0, it is not available. Lastly, the heading specific software is instructed to initialise the heading sensor.

'Inithdg' is called by 'init'.

### 4.2.9.2 'Gethdgdata( var hdgdata: hdgdatatype);'

This procedure is here to keep up the hierarchy. It only calls the procedure 'collecthdg'.

'Gethdgdata' is called by 'getdata'.

### 4.2.9.3 'Exechdgcommand( command: commandtype);'

This procedure is here to keep up the hierarchy. It only calls the procedure 'exechdgtxcommand'.

'Exechdgcommand' is called by 'execcommands'.



#### **4.2.9.4 'Closehdg( hdgdata: hdgdatatype);'**

This procedure is here to keep up the hierarchy. It only calls the procedure 'closehdgtx'.

'Closehdg' is called by 'closedown'.

#### 4.2.10 'DGPS'

This unit provides routines specially written for DGPS support. These routines are written by Peter Vianen, Maarten uit de Haag and Marco Meijer. Special MLS auxiliary data words are decoded to extract DGPS information. When DGPS information is decoded, the GPS correction is calculated using a first order approximation as a function of time.

By using the compiler directives '{\$N+,E+}', the compiler is instructed to compile for a mathematic coprocessor if present. If it is not present, an emulating library is be included.

Input	:	-
Output :	-	
Used by	:	'mias'
Uses	:	'miasglob' and 'miscell'

##### 4.2.10.1 'Inltdgps( var dgpsdata: dgpsdatatype);'

This procedure initialises the DGPS part of MIAS. This means that the specific DGPS variables are cleared.

'Inltdgps' is called by 'init'.

##### 4.2.10.2 'Getdgpsdata( var mlsdata: mlsdatatype; var dgpsdata: dgpsdatatype);'

This procedure extracts the specific DGPS ADW's from the MLS data. If no valid MLS data is available, the procedure is exit. Because the MLS ADW's B and C are not assigned yet by ICAO, the contents of these words are stored as bits. It has not yet been decoded.

Because only ADW C1 words contain DGPS information, the ADW C is checked to see if the address is 1. If not, the procedure is exit. If the ADW is valid, the satellite identification and the GPS time of transmission is read, as well as the correction parameters.

'Getdgps' is called by 'getdata'.

The following procedures and functions are included in procedure 'getdgpsdata'.

#### **4.2.10.2.1 'Adw\_read( adw: longint; position, bits: integer): longint;'**

This function decodes part of a longint 'adw'. 'Adw' is treated as a bit string. The 'bits' bits starting at 'position' are converted to a longint which is returned. The first bit one is the MSB.

'Adw\_read' is called by 'Getdgpsdata'.

#### **4.2.10.2.2 'Twos\_complement( adw: adwtype; position, bits: integer): longint;'**

This function decodes part of a longint 'adw'. 'Adw' is treated as a two's complement bit string. The 'bits' bits starting at 'position' are converted to a longint with a sign according to the two's complement rules, which is returned.

'Twos\_complement' is called by 'getdgpsdata'.

### **4.2.10.3 'Calcdgps( var alldata: alldatatype);'**

This procedure uses the collected DGPS data to calculate the DGPS pseudo range correction that is valid for the current time. The delay from the DGPS transmission to the aircraft reception, consists of two parts: the delay due to the DGPS correction calculation in the DGPS reference station, and the delay used for transmission and acquisition of data. If DGPS data is too old, the data is not used any more.

'Calcdgps' is called by 'calcpas'.

### **4.2.10.4 'Execdgpscommand( command: commandtype);'**

This procedure is here only for the hierarchy. It does nothing.

'Execdgpscommand' is called by 'execcommands'.

#### **4.2.10.5 'Closedgps( dgpsdata: dgpsdatatype);'**

This procedure is here only for the hierarchy. It does nothing.

'Closedgps' is called by 'closedown'.

## 4.2.11 'USER'

This unit is meant to interface the MIAS program with the operator. 'Key\_cons' is used to scan the keyboard and display messages on the display. This unit is only for keeping up the hierarchy. It only calls other procedures.

Input	:	-
Output :	:	-
Used by	:	'mias'
Uses	:	'miasglob' and 'key_cons'

### 4.2.11.1 'Inituser;'

This procedure only calls 'openin\_outputdev'.

'Inituser' is called by 'init'.

### 4.2.11.2 'Sendusermessage( message: commandtype);'

This procedure only calls 'sendmessage'.

'Sendusermessage' is called by 'init', 'exccommands' and 'sendposition'.

### 4.2.11.3 'Getusermessage( var message: commandtype);'

This procedure only calls 'getmessage'.

'Getusermessage' is called by 'getusercommands'.

### 4.2.11.4 'Senduserflags( message: commandtype);'

This procedure only calls 'sendflags'.

'Senduserflags' is called by 'dispflags'.

#### **4.2.11.5 'Saveequipmentmessage( message: commandtype);'**

This procedure only calls 'savemessage'.

'Saveequipmentmessage' is called by 'collectgpsrec', 'collectmlsrec', 'collectatt', 'collecthdg' and 'calchybridpos'.

#### **4.2.11.6 'Closeuser;'**

This procedure only calls 'closein\_outputdev'.

'Closeuser' is called by 'closedown'.

#### 4.2.12 'POSCALC'

This unit provides procedures to calculate the user position using MLS, GPS, attitude and heading information. This unit was written by René van Leeuwen described. The rest of the unit is described in [3].

By using the compiler directives '{\$N+,E+}', the compiler is instructed to compile for a mathematic coprocessor if present. If it is not present, an emulating library is included.

'Poscalc' also declares a variable 'mlsantposition', which is global in the unit. It contains the MLS antenna position, which is in principle different from the desired position. The desired position is the position of the landing gear or the centre of gravity (CG).

Input	:	MLS, GPS, attitude and heading and antenna vectors
Output :		Position of a desired point on the aircraft
Used by	:	'mias'
Uses	:	'miasglob', 'matrix' and 'mathx'

##### 4.2.12.1 'Calchybridpos( var alldata: alldatatype; allowed\_error: double; var position: positiontype);'

This procedure calculates the user position in a hybrid way, using GPS, MLS attitude and heading information. The variable 'allowed\_error' contains the error allowed in the position iteration. This is not the position accuracy!! As soon as a valid position is found, the 'position.flag' field is set to 'false'. Also the 'position.EcefTrueLocalFalse' field is set to the right value.

'Calchybridpos' is called by 'calcpas'.

##### 4.2.12.2 'Convert\_pos\_to\_ecef( var position: positiontype);'

This procedure converts the WGS-84 position in latitude, longitude and altitude to ECEF coordinates. It uses the WGS-84 fields from the variable 'position' and fills the ECEF fields. This procedure was written by G.L. van Eendenburg and edited by Marco Meijer to conform to specific needs.

'Convert\_pos\_to\_ecef' is called by 'init'

#### **4.2.12.3 'Convert\_pos\_to\_wgs( var position: positiontype);'**

This procedure converts the position in ECEF coordinates to latitude, longitude and altitude coordinates, all in WGS-84. It uses the ECEF fields from 'position' and fills in the WGS-84 fields. This procedure was written by G.L. van Eendenburg and edited by Marco Meijer to conform to specific needs.

'Convert\_pos\_to\_wgs' is not called.

Note: This procedure is here only for completeness.



### 4.2.13 'GPSCALC'

This unit provides routines for calculating the satellite position as function of time and corrections of the transmission time for satellite clock errors, relativistic effects and ionospheric, tropospheric effects. The unit is meant for single frequency users using the L1 frequency (SPS).

The unit instructs the compiler to use a coprocessor if present or to start emulating the coprocessor.

Some constants are declared for use in the procedures. They represent properties of the earth and the universe. Also the number pi and the L1 frequency are declared. The global variables declared are for use with DGPS calculations.

Input	:	-
Output :	-	
Used by	:	'gps'
Uses	:	'miasglob', 'gpsglob' and 'mathx'

#### 4.2.13.1 'Clockcorrection( sv: byte; var gpsint: gpsinttype);'

This procedure corrects for the satellite clock errors. It should be used in DGPS mode also. The procedure uses the clock parameters from the GPS message to correct the satellite clock. More information can be found in [2].

'Clockcorrection' is called by 'calcgps'

#### 4.2.13.2 'Relcorrection( sv: byte; var gpsint: gpsinttype);'

This procedure corrects for relativistic effects. It should be used in DGPS mode also. It uses parameters from the GPS message. More information can be found in [2].

'Relcorrection' is called by 'calcgps'.

#### **4.2.13.3 'L1correction( sv: byte; var gpsint: gpsinttype);'**

This procedure corrects for group delay effects on the L1 frequency. It should not be used in DGPS mode. This procedure only adds a factor from the GPS message to the transmission time. More information can be found in [2].

'L1correction' is called by 'calcgps'.

#### **4.2.13.4 'Ionosphericcorrection( sv: byte; var gpsint: gpsinttype);'**

This procedure corrects for delay effects due to the ionosphere. It should not be used in DGPS mode. The ionospheric correction is calculated using the user position and the azimuth and elevation angle from the GPS receiver to the selected satellite. More information can be found in [2].

Note: Use 'clockcorrection', 'l1correction' and 'elev\_azim' before using this procedure, because the transmission time, elevation and azimuth may not be correct when using this procedure for the first time.

'Ionosphericcorrection' is called by 'calcgps'

#### **4.2.13.5 'Troposphericcorrection( sv: byte; var gpsint: gpsinttype);'**

This procedure corrects for delay effects due to the troposphere. It should not be used in DGPS mode. This procedure uses the height above Mean Sea Level (MSL) and the elevation of the satellite above the horizon to compute the delay due to the troposphere. More information can be found in [2].

Note: Use 'elev\_azim' before using this procedure.

'Troposphericcorrection' is called by 'calcgps'

#### **4.2.13.6 'Eccentricanomaly( mk: double; var gpsint:gpsinttype; sv: byte): double;'**

This function calculates the eccentric anomaly of the satellite orbit. It uses 'Mk', which is the mean anomaly, as a first estimate for an iteration. Also the eccentricity of the satellite orbit is used. These parameters are supplied by the GPS message.

This procedure is iterative and is stopped if the difference between the last estimates of 'Ek' (eccentric anomaly) is smaller than 'epsilon' ( $=1E-9$ ) or the number of iterations is greater than 'maxiter' ( $=20$ ). More information can be found in [2].

'Eccentricanomaly' is called by 'svposition' and 'relcorrection'

#### **4.2.13.7 'Svposition( sv: byte; var gpsint: gpsinttype);'**

This procedure calculates the satellite position in ECEF coordinates. In this procedure first the time is calculated for which the ephemeris parameters are valid. The time is corrected for end/begin of week transitions. Then the coordinates of the satellite are calculated. For the 'arctan' function there is need for some protection to prevent an overflow. The result of the 'arctan' can be misinterpreted, so the quadrant of the answer is calculated using the argument again. More information can be found in [2].

'Svposition' is called by 'calcgps'

#### **4.2.13.8 'SVpos\_earthadjusted( sv: byte; Var gpsint: gpsinttype);'**

This procedure is meant to correct the satellite position, because it rotated a little between the transmission and reception of the signals. The travel between reception and transmission is calculated using the pseudo range and the GPS receiver clock offset. Then the anglur rotation is calculated by multiplying the travel with the earth's rotation rate. After that the satellite position (X,Y,Z) is transformed to the new position.

'SVpos\_earthadjusted' is called by 'calcGPS'.

#### **4.2.13.9 'Elev\_azim( sv: byte; var gpsint: gpsinttype; position: positiontype);'**

This procedure calculates the elevation and azimuth angles (rad) of a specific satellite relative to the GPS receiver and the local horizontal.

Note: Be sure that the position which is input has valid WGS-84 and ECEF coordinates which match each other.

'Elev\_azim' is called by 'calcgps'

#### **4.2.13.10 'Conv\_pos\_to\_wgs( var position: positiontype);'**

This procedure converts the ECEF coordinates to latitude, longitude and altitude (WGS-84) coordinates . It uses the ECEF fields from the variable 'position' and fills the WGS-84 fields. This procedure was written by G.L. van Eendenburg and edited by Marco Meijer to conform to specific needs.

'Conv\_pos\_to\_wgs' is called by 'calcgps'

#### **4.2.13.11 'Calc\_pr( sv: byte; var gpsint: gpsinttype);'**

This procedure calculates the pseudo range from the transmission and reception time. The integrated carrier phase should be added to the transmission time before this procedure is used.

'Calc\_pr' is called by 'calcgps'

#### **4.2.13.12 'Calcsmoothpr( sv:byte; var gpsint: gpsinttype);'**

This procedure smooths the pseudoranges from the GPS receiver. It is meant to be used only in the DGPS reference station. It is written by Peter Vianen.

Note: No interface for this procedure is defined yet.

## 4.2.14 'MATHX'

This unit contains some mathematical functions that are not supplied by Turbo Pascal. The unit starts with instructing the compiler to use the coprocessor or to use an emulating library. The constant pi is declared too.

Input	:	-
Output :	-	
Used by	:	'gpscalc'
Uses	:	'miasglob'

### 4.2.14.1 'Tan( arg: double): double;'

This procedure calculates the tangent of the argument by dividing the sinus and the cosines of the argument. If the cosines is zero, an overflow might occur. This is protected by assigning 1E38 or -1E38 to the answer.

'Tan' is called by 'troposphericcorrection'.

### 4.2.14.2 'Arccos( x: double): double;'

This procedure calculates the arccosines of the argument 'x'. The correct quadrant is calculated using the sign of the argument. Arguments that are too big are limited to 1 or -1. The arccosines is calculated using the Pythagoras theorem and the arctangent.

In a circle with radius 1, the argument 'x' represents the 'x' coordinate of a point at the circle at an angle of  $\arccos(x)$ . Using Pythagoras theorem, the y coordinate can be found, which is  $\sqrt{1-x^2}$ . Then using the 'arctan' function, the angle  $\arccos(x) = \arctan(\sqrt{1-x^2}/x)$ . If the argument 'x' is negative, the pi should be added to the calculated angle to get the right quadrant (only quadrant 1 and 2 can be used here).

'Arccos' is called by no function.

#### **4.2.14.3 'Arcsin( x: double): double;'**

This procedure calculates the arcsines of the argument 'x'. The correct quadrant is calculated using the arctangent function. Arguments that are too big, are limited to 1 or -1. The arcsines is calculated using the theorem of Pythagoras and the arctangent function. See 'arccos'.

'Arcsin' is called by 'elev\_azim'.

## 4.2.15 'MATRIX'

This unit contains routines to handle matrices. The unit was written by Borland International and is part of the Numerical Methods Toolbox. This unit has an 'include' compiler directive, which instructs the compiler to include a file called 'matrix.inc'.

A slight change is made in the toolbox: The compiler is instructed directly to use the coprocessor and to use an emulating library if the coprocessor is not available. The 'miasglob' unit is used to define the 'double' type.

In the following, only the procedure that is used in the MIAS program is mentioned.

Input	:	-
Output :	-	
Used by	:	'poscalc'
Uses	:	'miasglob'

### 4.2.15.1 'Inverse( dimen: integer; data: tnmatrix; var inv: tnmatrix; var error: byte);'

This procedure calculates the inverse of the matrix 'data'. The matrix 'data' must be square. The dimension of the matrix 'data' is given in 'dimen'. The result is given in 'inv'. The variable 'error' gives a status indication.

'Inverse' is called by 'calc\_weighed\_leastsqmatrix'

## 4.2.16 'GPSENGINE'

This unit provides interfacing between the main program and the GPS Engine or a MX4200D, which are Magnavox products. The GPS Engine and the MX4200D are very similar products. The MX4200D receiver uses has a antenna connector and a 25 pin female D connector (the Engine has two 25 pin female D connectors), which is used for data transmission from the receiver to a computer and vice versa. The D-25 connector contains four output ports (the Engine has two ports), from which only two ports are used. One port is called 'port 0' or 'port B' the other 'port 1' or 'port C'. The MX 4200D ports are called 'B' and 'C' and signal their information in RS 422. The GPS Engine signals in RS 232. 'Port 0' outputs position and control information. 'Port 1' outputs raw data measurements and can receive DGPS error corrections in RTCM SC-104 format. The ports are connected to two RS 422 comports (the Engine to two RS 232 comports).

By using the compiler directives '{\$N+,E+}', the compiler is instructed to compile for a mathematic coprocessor if present. If it is not present, an emulating library is be included. The unit also declares some global variables and constants.

At initialisation, the array 'two2power' is filled with the series 1 to  $2^{55}$ . The range is chosen to handle the demand from the ephemeris parameters. The array 'two2power' is used as a look-up-table, which is faster than calculating  $2^x$  every time it is needed.

Input	:	-
Output	:	-
Used by	:	'gps'
Uses	:	'miasglob', 'gpsglob', 'user', 'miscell', 'com_4' and 'crt'

Note: The information in this section is for the GPS Engine and the MX4200D receiver.

### 4.2.16.1 'Initgpsrec( var error: boolean);'

This procedure will initialise the GPS receiver, in this case the GPS Engine or MX4200D receiver. First it will open the configuration file 'mias.cfg' and extract the value for the following variables:

port0	{ comport number for 'port 0' or 'B' information}
port1	{ comport number for 'port 1' or 'C' information}
completeinfo	{ 1: pseudo ranges are output when all tracked pseudo ranges are available; 0: also incomplete info is passed}



Then the comports are initialised. Both comports are initialised to communicate at 4800 Baud, 8 bits, 1 stop bit and no parity.

'Initgpsrec' is called by 'initgps'.

#### **4.2.16.2 'Collectgpsrec( var gpsint: gpsinttype);'**

This procedure will call two other procedures, one for 'port 0' and one for 'port 1'. It will call these procedures a number of times to empty the comport buffer, depending on the initial value of the 'counter' variable.

'Collectgpsrec' has two internal procedures: 'collectport0' and 'collectport1', which will be discussed now.

'Collectgpsrec' is called by 'getgpsdata'.

##### **4.2.16.2.1 'Collectport0( var gpsint: gpsinttype);'**

This procedure will collect messages from 'port 0' of the GPS Engine or MX4200D receiver. It will then convert them to the associated pascal variable. To make sure that only complete information is output to the 'GPS' unit, the variable 'tempGPSint' is used. 'TempGPSint' contains all valid information. In case the 'GPSint' variable was erased in the 'GPS' unit, the 'tempGPSint' variable will fill in the right information again.

Note: Only 'collectport0' and 'collectport1' can store information in 'gpsint'.

To retrieve a message from the 'com' port buffer, the program checks if the number of characters in the buffer is greater or equal to 80. Because the maximum length of a message is 80, even the biggest message can be retrieved successfully. If the first character read is the start of a message, the rest of the message is read too. The end of a message is indicated by a 'LineFeed' (LF). If a complete message was read, the message is checked if it has a valid header. If the header is ok, the record number is retrieved from the message, and the appropriate procedure will be called to extract more information from the message. If no valid information is read, the procedure is exit.

If the first character read was not the beginning of a message, the procedure will read more characters from the buffer until a LF was found or no more characters are in the buffer. The last action of the procedure is to update 'tempGPSint'.

Valid messages are displayed on the screen and save to disc

'Collectport' has 1 internal procedure: 'statusreport' which is described shortly.

'Collectport0' is called by 'collectgpsrec'.

#### 4.2.16.2.1.1 'Statusreport( rec: string; var gpsint: gpsinttype);'

This procedure will use 'port 0' message number 0, to extract the information from the GPS receiver about number of satellites that are being tracked. If no valid number could be determined, a '-1' is assigned to the variable 'gpsint.numofsat'.

'Statusreport' is used by 'collectport0'.

#### 4.2.16.2.2 'Collectport1( var gpsint: gpsinttype);'

This procedure does essentially the same as 'collectport0', but now for 'port 1' instead of 'port 0'. The variable 'tempGPSint' is used to initialise the 'gpsint' variable. 'Port 1' messages also have maximum length of 80 characters. The beginning of a message is more difficult to find here. A better technique is to find the end of a previous message. So the procedure reads a character and checks if it is a LF. If it is, the next characters read will form a message. The LF at the end of this message should not be read, so that the next message will be read correctly.

The message number is determined and the appropriate procedure is called. If the first character read was not a LF, the characters from the buffer are read until a LF was found. (This LF is not read from the buffer!) As soon as all information is extracted from the message, the 'tempGPSint' variable is updated.

Then a tricky part comes. For the rest of the program it is sometimes necessary, that only complete GPS information be output. That is: all pseudo ranges are output and not one or two (if there are more pseudo ranges). But on the other hand, invalid (timed out) information is not wanted. That is why the current system time is compared to the time of the first pseudo range of an ensemble ('Tpr'). The

difference between those times should be no more than 1 second. (The GPS Engine and MX4200D have an update rate of 1 second).

If the pseudo ranges are timed out or all pseudo ranges from an ensemble are available, the information is output and the pseudo range flag fields of 'tempGPSint' variable are cleared, so it can't output the same information (old information) twice. Also the 'svcount' variable is cleared. It contains the number of pseudo ranges read from an ensemble until now.

If the pseudo ranges are not timed out or not all pseudo ranges from an ensemble are read yet, the 'complete\_info' variable decides what will happen. ('complete\_info' was read from the configuration file.) If 'complete\_info' is 1, nothing will be output (clear the fields in 'gpsint'). If 'complete\_info' is 0, all available information is output.

'Collectport1' includes the following procedures: 'Conv\_ascii\_2\_val', 'scale', 'twoscomplement', 'rawdata', 'ionoscor', 'get\_sv\_id', 'clockinfo' and 'ephemeris'.

Note: Valid messages are displayed on the screen and saved to disc.

'Collectport1' is called by 'collectgpsrec'.

#### 4.2.16.2.2.1 'Conv\_ascii\_2\_val( rec: string; var temp: temptype; var error: boolean);'

This procedure converts a character string containing hexadecimal data, from the GPS engine receiver to an array containing the values represented by the pairs of hexadecimal numbers. In the string are 24 pairs of hexadecimal numbers.

'Conv\_ascii\_2\_val' is called by 'ionoscor', 'clockinfo' and 'ephemeris'.

Note: This routine can be shorter by using the procedure 'Val' on a string with '\$xx' instead of 'xx'.

Where 'xx' is the hexadecimal number to convert.

#### 4.2.16.2.2.2 'Scale( temparray: arraytype; pointer, startbit, nr\_of\_bits: byte):double;'

This function will calculate a value from the bits that are represented as decimal numbers in 'temparray'. The first bit used is bit number 'startbit' in array element 'pointer' from 'temparray'. The 'nr\_of\_bits' indicates the number of bits that form the desired number.

Note: This function will only work with the 'nr\_of\_bits' variable is less or equal to 32.

'Scale' is called by 'ionoscor', 'clockinfo' and 'ephemeris'.

#### 4.2.16.2.2.3 'Twoscomplement( number: double; nr\_of\_bits: byte):double;'

A binary number was converted to the decimal 'number' using the natural binary code. This binary number was a two's complement number. This function decodes 'number' with the natural binary code and encodes it again using the two's complement code. If the MSB indicated by the 'nr\_of\_bits' variable is 0, then the two's complement number is the same as the binary number. If the MSB is 1, then the two's complement number can be found by:  $-2^{nr\_of\_bits} + number$ .

'Twoscomplement' is called by 'ionoscor', 'clockinfo' and 'ephemeris'.

#### 4.2.16.2.2.4 'Rawdata( rec: string; var gpsint: gpsinttype; var svcount: shortint);'

This procedure will extract pseudo range information from the GPS Engine or MX4200D record 1 from the raw data output 'port 1'. It will extract the satellite PRN number, the user time, transmission time, integrated carrier phase and the raw code offset.

If the raw data currently handled is the first of a new ensemble of pseudorange record ('old\_user\_ms'  $\neq$  'user\_ms'), the information is put back in the comport buffer, to avoid conflicts. In that case 'old\_user\_ms' becomes 'user\_ms', so the next time the message is read, it will be processed. At the same time, the 'svcount' variable becomes 'numofsat', so 'collectport1' will know that all available pseudo range information is read. The last action here, is to store the current time in 'Tpr', so that a time out will be given 'Valid\_Tpr' seconds after 'Tpr'.

If the raw data currently handled is not the first of a new ensemble of pseudo range records, the transmission time and integrated carrier phase are calculated and stored in 'GPSint'. The flag for the satellite, from which the pseudo range came, is set to 'false', to indicate a valid measurement and the 'svcount' is incremented.

'Rawdata' is called by 'collectport1'.

#### 4.2.16.2.2.5 'Ionoscor( rec: string; var gpsint: gpsinttype);'

Ionospheric parameters are passed through by the GPS Engine or MX4200D receiver by raw data 'port 1', message number 135. This procedure uses a string, with ASCII characters representing hexadecimal numbers, and converts it to the ionospheric correction parameters as specified in the [2]. First the string is converted to a series of numbers using 'conv\_ascii\_2\_val'. Then the parameters are calculated using 'scale' and 'twoscomplement'.

The time the ionospheric correction parameters came in is stored in 'gpsint.tionos'. The ionospheric corrections are valid for 'Valid\_Tionos'.

'Ionoscor' is called by 'collectport1'.

#### 4.2.16.2.2.6 'Get\_sv\_id( rec: string; var sv\_id: byte);'

Ephemeris information comes in 4 different messages, which belong together. The first message gives the satellite number. The next messages give the parameters. This procedure extracts the satellite number from the raw data message 200. This number is then stored in 'sv\_id'. If no valid satellite number could be decoded, 'sv\_id' will be 0. The current time is stored in 'Tsv\_id'. This is done to make old (timed out) information invalid.

'Get\_sv\_id' is called by 'collectport1'.

#### 4.2.16.2.2.7 'Clockinfo( rec: string; sv\_id: byte; var gpsint: gpsinttype);'

The procedure will extract the satellite clock error information from raw data output record 201. First the 'sv\_id' variable is checked for a possible time out, using 'Tsv\_id' and 'valid\_tsv\_id'. If 'sv\_id' is 0, 'sv\_id' is invalid and the procedure is exit. The ASCII string is converted to values using 'conv\_ASCII\_2\_val'. Then the values are assigned to the fields of 'gpsint.prn[sv\_id].clock' using 'scale' and 'twoscomplement'. Also the health information from message 201 is used. And finally, the current time is stored in 'Tck', because the clock information is only valid for 'Valid\_Tck'.

'Clockinfo' is called by 'collectport1'.

#### 4.2.16.2.2.8 'Ephemeris( recnum: integer; rec: string; sv\_id: byte; var gpsint: gpsinttype);'

This procedure will use the ephemeris information in 'rec' to calculate the ephemeris parameters and store them in 'gpsint.prn[sv\_id].ephemeris'. First the validity of 'sv\_id' is checked. It is checked for time out and invalid value.

If no errors occurred, the ASCII information in 'rec' is converted to numbers using 'conv\_ascii\_2\_val'. Depending on the message number ('recnum'), the appropriate parameters are initialised using 'scale' and 'twoscomplement'.

The 'IODE' word is special. It should be the same for message number 202 and 203. It is transmitted in both subframes 2 and 3 of the GPS message. If the IODE is different, the ephemeris parameters in message 202 and 203 do not belong together and are therefore invalid.

When the information is valid, the current time is stored in 'Tephem'. The ephemeris data is valid for 'Valid\_Tephem'. The last action is to make 'sv\_id' 0, which makes 'sv\_id' invalid. So 'sv\_id' can't be used when it is not valid any more.

'Ephemeris' is called by 'collectport1'.

#### 4.2.16.3 'Execgpsrecommand( command: commandtype);'

This procedure translates the commands from the rest of the program, which are in a general format, to specific GPS Engine or MX4200D commands. The procedure recognises several commands: 'RESET', 'INIT' and 'SEND EPHEMERIS ETC'. Of course more commands are possible.

The syntax of 'Reset' is simple: 'RESET'. 'Reset' commands the GPS Engine or MX4200D to execute a 'Tepid' start, which means that if the receiver has an almanac, it will skip the search-the-sky phase and start acquiring satellites immediately. The GPS Engine MX4200D command is: '\$PMVXG,018,T'.

Note: 'Reset' is disabled now, because using 'reset' when the receiver power has just been turned on, will cause the receiver to stop sending any info, appearing to be 'dead'.

The syntax of 'Init' is more difficult: 'INIT DD MM YY HHMM DDMM.MMMM N DDMM.MMMM E HHHHH.H AA.A EL'. From left to right:

MM means	Months in the year.
YY means	Years in the 20 th century.

HHMM means	Hours and minutes in the day.
DDMM.MMMM means	latitude in Degrees and Minutes plus fraction of a minute.
N means	N for North or S for South.
DDMM.MMMM means	longitude in Degrees and Minutes plus fraction of a minutes.
E means	E for East of W for West.
HHHHH.H means	Height above the geoid.
AA.A means	the horizontal acceleration factor. (For tracking loop bandwidth.)
EL means	is the elevation mask angle.

The 'init' command transmits an initialising string (GPS Engine command: \$PMVXG,000,...'), a navigation control string (GPS Engine command: \$PMVXG,001,...'), a data control string for the raw data port (GPS Engine command: \$PMVXG,024,...'), a time recovery string (GPS Engine command: \$PMVXG,023,...') and data control strings for the control port (GPS Engine command: \$PMVXG,007,...').

Note: The GPS Engine initialising command seems to have no (great) effect on the receiver.

Note: The height input with the 'init' command is the height above the geoid, but the height in the GPS receiver is the height above the Means Sea Level.

The syntax of 'send ephemeris etc' is simple: 'SEND EPHEMERIS ETC'. The GPS receiver command is: '\$PMVXG,027,...'.

'Execgpsrecommand' is called by 'execgpscommand'.

#### **4.2.16.4 'Closegpsrec;'**

This procedure puts the 'com' ports back to normal and saves some specific GPS Engine or MX4200D parameters in the configuration file. These are the same parameters as were read from the configuration file.

'Closegpsrec' is called by 'closegps'.

## 4.2.17 'MLSBENDIX'

This unit provides routines to interface with the Bendix MLS-20A receiver. This receiver outputs the received angles in a ARINC 429 like format. This means, that it uses the same transmission technique (bipolar return to zero with a bit rate of 100 Kbit per second), but it defines its own word structure. Fortunately, the address field is the same as in ARINC 429. The labels for specific functions differ though.

The data words (basic and auxiliary) are output on a separate bus, but cannot be read with an ARINC 429 interface, because they are not structured in words. That is why the data words can be simulated by hard coding the data words that are appropriate for the flight tests at Schiphol Airport. A special B/ADW card is available now to read the decoded DPSK output of the receiver. It interrupts the program only when valid function preambles are received.

Also DGPS information is received here, because it is coded as MLS Auxiliary data words.

The compiler is instructed to use a coprocessor or an emulating library. Some type, constants (ARINC 429 labels) and variables are declared. At initialisation, the variables 'adw\_a', 'adw\_b', 'adw\_c', 'mlsint', 'Tangle', 'Valid\_Tangle' and 'anglebegin' are declared.

Note: Although a part of the software for the B\ADW card is in this unit, it should be better to give this part a separate unit. It is in fact a driver.

Input	:	-
Output :		-
Used by	:	'mls'
Uses	:	'miasglob', 'mlsglob', 'crt', 'ar429comm', 'ar429', 'miscell' and 'user'.

### 4.2.17.1 'Initmlsrec( var error: boolean);'

This procedure initialises the MLS receiver. First the relevant variable values are read from the configuration file. These values are listed under the header 'MLSBENDIX'. 'completeinfo', 'IRQ', and 'cardaddress' are updated. 'Completeinfo' is used to output all information that belong together (Azimuth and Elevation) or all information that is present at a certain time. 'IRQ' is the interrupt number of the B/ADW card. 'Cardaddress' is the address of the B/ADW card.

Then the communication link for DGPS is setup. This link will work with a comport and a FSK modem. This is a temporary solution until the ADW's can be programmed dynamically in the MLS



transmitter at Schiphol. Then a table is prepared containing the ARINC 429 labels (in octal) which the ARINC-429-to-PC interface should pass through. Then the ARINC-429-to-PC interface (M429PC) is initialised. Finally the BDW and ADW preamble are programmed in the B/ADW card and the B/ADW card is started.

'Initmlsrec' is called by 'initmls'.

#### **4.2.17.2 'Collectmlsrec( var mlsint: mlsinttype);'**

This procedure gets the information from the MLS receiver and updates the relevant data fields. The variable 'tempMLSint' contains all the valid MLS information until now. To make sure no information is overwritten outside 'CollectMLSrec', 'tempMLSint' will be copied to 'MLSint'. Then an ARINC 429 word from the data link for DGPS is received by calling 'ar429comm.getar429word'. If the word is valid, it will be written on the screen and saved to disc. Then a 'case' statement is entered, which takes the appropriate action depending on the label of the received word. The same is done for ARINC 429 words from the ARINC-429-to-PC interface.

After these actions, the FIFO buffer from the B/ADW card is emptied. Every message starts with FF FF<sub>HEX</sub>. If this header is read, a new line starts and the previous message is complete. It is therefore stored to file and the 'badwline', which contains the message, is cleared. If the bytes read were not the header, they are added to the bytes which came in earlier. The 'wordready' boolean indicates if a message is complete. If the message is complete, the byte counter 'bytecount' decides what data word is received: BDW or ADW. Depending on the data word, the bits from the received bytes are calculated and copied to a variable 'ar429word' for the BDW's. Then BDW information is extracted from 'ar429word' and 'bytecount' and 'wordready' is reset. The ADW bytes are converted to bits and copied to 'ADword'. If the parities are okay, the ADW information is extracted from the 'ADword' and 'bytecount' and 'wordready' are reset.

Now 'mlsint' is copied to 'tempMLSint', to keep it up to date. The next statements are for test using a MLS receiver which does not output the BDW's and ADW's. The fields of the BDW's and ADW's in 'MLSint' are assigned with constants which are declared in 'MLSglob'.

If both azimuth and elevation angles are transmitted, and only one of them is received, the 'anglebegin' variable is set to 'true'.

At this point, it should be decided if the procedure should output information. If there is still information that was not output yet and it is timed out, or there won't be other information

('anglebegin=false') then the information is output in 'MLSint'. To prevent information from being output twice (invalid), the variable 'tempMLSint' is cleared.

If there was not time out and 'anglebegin=true', 'completeinfo' decides what to do. If it is 0, all available information is output. If it is 1, only a complete ensemble of information is output.

'CollectMLSrec' contains 22 internal procedures and functions: 'adw\_present', 'hamming\_fail', 'address\_fail', 'adw\_address', 'conv\_adw', 'adw\_a\_conv', 'adw\_a\_coll', 'adw\_b\_conv', 'adw\_b\_coll', 'adw\_c\_conv', 'adw\_c\_coll', 'conv\_bas', 'bas\_1\_conv', 'bas\_2\_conv', 'bas\_3\_conv', 'bas\_4\_conv', 'bas\_5\_conv', 'bas\_6\_conv', 'el\_conv', 'az\_conv', 'discretes\_conv' and 'hex2string'.

'CollectMLSrec' is called by 'getmlsdata'.

#### *4.2.17.2.1 'Adw\_present( adw\_pres: prestype):boolean;'*

This function returns 'true' if all fields of 'adw\_pres' contain 'true'. Which indicates that all ARINC 429 words that form a specific ADW are present. It returns 'false' if this is not true. This function was written by Maarten uit de Haag.

'Adw\_present' is called by 'adw\_a\_conv', 'adw\_b\_conv' and 'adw\_c\_conv'.

#### *4.2.17.2.2 'Hamming\_fail( adw:adwtype):boolean;'*

This function checks the Hamming code which protects the ADW for errors. More information about his Hamming code can be found in [1, p. 150B]. This function was written by Maarten uit de Haag and revised by Marco Meijer.

The values for the 6 parity bits are computed and compared with the actual parities. If they are equal, a 'false' is output to indicate a failure. Else a 'true' is output to indicate success.

'Hamming\_fail' is called by 'adw\_a\_conv', 'adw\_b\_conv' and 'adw\_c\_conv'.

#### 4.2.17.2.3 *'Address\_fail( adw: adwtype): boolean;'*<sup>1</sup>

This procedure checks the address code of the ADW. This address code is protected with 2 parity bits. More information about this can be found in [1, p 150B]. This function was written by Maarten uit de Haag and revised by Marco Meijer.

This function returns 'true' to indicate a failure and 'false' to indicate a success.

'Address\_fail' is called by 'adw\_a\_conv', 'adw\_b\_conv' and 'adw\_c\_conv'.

#### 4.2.17.2.4 *'Adw\_adress( adw: adwtype):byte;'*

This function takes bits 13 to 18 of the ADW and translates them to a decimal address. For more information, see [1, p 150]. This function was written by Maarten uit de Haag and revised by Marco Meijer.

'Adw\_adress' is called by 'adw\_a\_conv', 'adw\_b\_conv' and 'adw\_c\_conv'.

#### 4.2.17.2.5 *'Conv\_adw( adw: adwtype; start: byte; number:byte):word;'*

This function takes some bits from the ADW and converts them to a number using the natural binary code. It takes the bits starting at bit number 'start' and it takes 'number' bits. The first bit number is 1. See [1, p 60CC] for more information. The LSB is output first.

'Conv\_adw' is called by 'adw\_a\_conv', 'adw\_b\_conv' and 'adw\_c\_conv'.

#### 4.2.17.2.6 *'Adw\_a\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype; a\_label: byte; var adw\_a\_pres: adw\_prestype; var adw\_a: adwtype);'*

This procedure takes specific bits from the ADW, converts them to a decimal number and assigns this number to the MLS variables that are represented by those bits. The variables are fields in 'MLSint'. More information can be found in [1, p 150A].

For every ADW, the reception time is stored to supervise time outs.

---

<sup>1</sup> The names of the procedures containing 'address' are consequently misspelled.

'Adw\_a\_conv' is called by 'Adw\_A\_coll'.

*4.2.17.2.7 ADW\_A\_coll( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype; a\_label: byte;  
Var ADW\_A\_pres: ADW\_prestype; Var ADW\_A: ADWtype);*

This procedure collects the ARINC 429 words that form the ADW A word. It fills in the 'ADW\_A\_pres' variable and takes the bits from the ARINC 429 words and puts them in 'ADW\_A'. If the ADW is complete and the parity are correct, the procedure 'ADW\_A\_conv' is called and the 'ADW\_A\_pres' variables are cleared to prevent an old ADW to be used.

'ADW\_A\_coll' is called by 'CollectMLSrec'.

*4.2.17.2.8 Adw\_b\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype; a\_label: byte; var  
adw\_b\_pres: adw\_prestype; var adw\_b: adwtype);'*

This procedure copies the ADW B from 'ADW\_B' to the 'AuxB'-field in 'MLSint' and records the time for a time tag.

'Adw\_b\_conv' is called by 'Adw\_b\_coll'.

Note: At this moment the ADW B words are not officially assigned yet.

*4.2.17.2.9 ADW\_B\_coll( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype; a\_label: byte;  
Var ADW\_B\_pres: ADW\_prestype; Var ADW\_B: ADWtype);*

This procedure collects the ARINC 429 words that form the ADW B word. It fills in the 'ADW\_B\_pres' variable and takes the bits from the ARINC 429 words and puts them in 'ADW\_B'. If the ADW is complete and the parity are correct, the procedure 'ADW\_B\_conv' is called and the 'ADW\_B\_pres' variables are cleared to prevent an old ADW to be used.

'ADW\_B\_coll' is called by 'CollectMLSrec'.

*4.2.17.2.10 'Adw\_c\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype; a\_label: byte;  
var adw\_c\_pres: adw\_prestype; var adw\_c: adwtype);'*

This procedure copies the ADW C from 'ADW\_B' to the 'AuxC'-field in 'MLSint' and records the time for a time tag.

'Adw\_c\_conv' is called by 'Adw\_c\_coll'.

Note: At this moment the ADW C words are not officially assigned yet.

*4.2.17.2.11 'ADW\_C\_coll( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype; a\_label:  
byte; Var ADW\_C\_pres: ADW\_prestype; Var ADW\_C: ADWtype);'*

This procedure collects the ARINC 429 words that form the ADW C word. It fills in the 'ADW\_C\_pres' variable and takes the bits from the ARINC 429 words and puts them in 'ADW\_C'. If the ADW is complete and the parity are correct, the procedure 'ADW\_C\_conv' is called and the 'ADW\_C\_pres' variables are cleared to prevent an old ADW to be used.

'ADW\_C\_coll' is called by 'CollectMLSrec'.

*4.2.17.2.12 'Conv\_bas( ar429word: ar429wordtype; start: byte; number: byte): word;'*

This function converts bits from 'Ar429word' to a decimal number. 'Start' indicates the starting position. 'Number' the number of bits to be read. The bit with number 'start' is the LSB.

'Conv\_bas' is called by 'bas\_1\_conv', 'bas\_2\_conv', 'bas\_3\_conv', 'bas\_4\_conv', 'bas\_5\_conv', 'el\_conv', 'az\_conv', 'baz\_conv' and 'discretes\_conv'.

*4.2.17.2.13 'Bas\_1\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);'*

This procedure takes bits from the ARINC 429 word containing Basic data word number 1. More information can be found in [1, p 146].

'Bas\_1\_conv' is called by 'collectmlsrec'.

*4.2.17.2.14 'Bas\_2\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);'*

This procedure takes bits from the ARINC 429 word containing Basic data word number 2. More information can be found in [1, p 146].

'Bas\_2\_conv' is called by 'collectmlsrec'.

*4.2.17.2.15 'Bas\_3\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);'*

This procedure takes bits from the ARINC 429 word containing Basic data word number 3. More information can be found in [1, p 146].

'Bas\_3\_conv' is called by 'collectmlsrec'.

*4.2.17.2.16 'Bas\_4\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);'*

This procedure takes bits from the ARINC 429 word containing Basic data word number 4. More information can be found in [1, p 146].

'Bas\_4\_conv' is called by 'collectmlsrec'.

*4.2.17.2.17 'Bas\_5\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);'*

This procedure takes bits from the ARINC 429 word containing Basic data word number 5. More information can be found in [1, p 146].

'Bas\_5\_conv' is called by 'collectmlsrec'.

*4.2.17.2.18 'Bas\_6\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);'*

This procedure takes bits from the ARINC 429 word containing Basic data word number 6. More information can be found in [1, p 146].

'Bas\_6\_conv' is called by 'collectmlsrec'.

#### 4.2.17.2.19 *'El\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);'*

This procedure takes the ARINC 429 word with elevation information and converts the elevation angle to the elevation Pascal variable. The sign is also checked. For more information, check [4]. Also the antenna in use and the reception time is stored.

Note: The information in [4] is incomplete: The angle are coded in two's complement. Negative numbers (as indicated by the sign bits) should be calculated as follows: Determine the value of the angle word with the natural binary code, subtract 4000<sub>HEX</sub> and multiply with 0.005 to obtain the angle in degrees.

'El\_conv' is called by 'collectmlsrec'.

#### 4.2.17.2.20 *'Az\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);'*

This procedure takes the ARINC 429 word with azimuth information and converts it to the azimuth Pascal variable. The sign is checked and left- or right clearances are checked. for more information, check [4]. Also the antenna in use and reception time is stored.

Note: See note in section 4.2.17.2.19.

'Az\_conv' is called by 'collectmlsrec'.

#### 4.2.17.2.21 *'Baz\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);'*

This procedure takes the ARINC 429 word with back azimuth information and converts it to the associated Pascal variable. The sign is checked and left or right clearances are checked. For more information, check [4]. Also the antenna in use and the reception time is stored.

Note: See note in section 4.2.17.2.19

'Baz\_conv' is called by 'collectmlsrec'.

#### **4.2.17.3 'Discretes\_conv( var mlsint: mlsinttype; ar429word: ar429wordtype);**

This procedure can receive an ARINC 429 word containing so called discretes. They are provided by an ARINC 727 receiver. This procedure converts the bits from the ARINC 429 word to the associated variable fields in 'mlsint'.

'Discretes\_conv' is called by 'collectmlsrec'.

##### **4.2.17.3.1 'Hex2string(x:byte):string;'**

This function converts a byte to its hexadecimal representation in ASCII characters. This function was written by Rob Luxen to store information from the B/ADW card to disc.

'Hex2String' is called by 'Collectmlsrec'.

#### **4.2.17.4 'Execmlsrecommand( command: commandtype);'**

This procedure executes commands from the program to the MLS receiver. It is possible to define commands here, which can be executed on all MLS receivers, like channel selection, self test, etc. These ASCII messages should be translated to another format. For the ARINC 727 receivers and the Bendix receiver, this can be the ARINC 429 format. The messages send are stored on disc.

Note: No commands are specified yet.

'Execmlsrecommand' is called by 'execmlscommand'.

#### **4.2.17.5 'Closemlsrec;'**

This procedure shuts down the communication with the MLS receiver by closing the serial communication for DGPS, the ARINC-429-to-PC interface and the B/ADW card. It also stores the 'completeinfo', the 'cardaddress' and the 'IRQ' variable in the configuration file.



## 4.2.18 'ATTBEAVER'

This unit contains the specific software needed to acquire the attitude (roll and pitch) from the Beaver. The unit does not play an important role in the MIAS program nor is it complicated.

The compiler is instructed to use a coprocessor or an emulating library. The value of pi is assigned and the pitch and roll offsets are declared. The unit uses software to read data from a synchro-to-digital converter.

Input	:	Angle in degrees
Output :		roll and pitch angles in radians
Used by	:	'att'
Uses	:	'miasglob', 'miscell', 'synchcnv', 'user' and 'crt'

### 4.2.18.1 'Initatttx( var error: boolean);'

This procedure opens the configuration file 'mias.cfg' and retrieves the pitch- and roll offsets from it. These numbers are important, because they define the zero pitch angle. (The Beaver is a tail wheel aircraft with a large pitch angle when parking.) Also the synchro-to-digital converter is initialised. This converter calculates the digital representation of the angle which is coded by the analog synchro signals.

'Initatttx' is called by 'initatt'.

### 4.2.18.2 'Collectatt( var attdata: attdatatype);'

This procedure gets data from the synchro-to-digital converter. The data represent the angles in degrees. The pitch and roll offsets are subtracted. The resulting angles are converted to radians. Also a message containing the read data is stored on disc.

'Collectatt' is called by 'getattdata'.

#### **4.2.18.3 'Execatttxcommand( command: commandtype);'**

This procedure is empty. Sending a command does not have any effect yet. This procedure is only here to keep up the hierarchy.

'Execatttxcommand' is called by 'execatt'.

#### **4.2.18.4 'Closeatttx;'**

This procedure returns the synchro-to-digital converter to its original condition. It also saves the pitch- and roll offsets to the configuration file.

'Closeatttx' is called by 'closeatt'.

#### 4.2.19 'HDGBEAVER'

This unit contains the specific software needed to acquire the heading of the Beaver. The unit does not play an important role in the MIAS program nor is it complicated.

The compiler is instructed to use a coprocessor or an emulating library. The value of pi is assigned and the heading offset is declared. The unit uses software to read data from a synchro-to-digital converter.

Input	:	angles in degrees
Output :		heading angle in radians
Used by	:	'hdg'
Uses	:	'miasglob', 'miscell', 'synchcnv', 'user' and 'crt'

##### 4.2.19.1 'Inithdgtx( var error: boolean);'

This procedure opens the configuration file 'mias.cfg' and retrieves the heading offset from it. This number is important, because it defines the zero heading angle. (Difference between true north and magnetic north.) Also the synchro-to-digital converter is initialised. This converter calculates the digital representation of the angle which is coded by the analog synchro signals.

'Inithdgtx' is called by 'inithdg'.

##### 4.2.19.2 'Collecthdg( var hdgdata: hdgdatatype);'

This procedure gets data from the synchro-to-digital converter. The data represents the heading angle in degrees. The heading offset is subtracted. The resulting angle is converted to radians. Also a message containing the read data is stored on disc.

'Collecthdg' is called by 'gethdgdata'.

##### 4.2.19.3 'Exechdgtxcommand( command: commandtype);'

This procedure is empty. Sending a command does not have any effect yet. This procedure is only here to keep up the hierarchy.

'Exechdgtxcommand' is called by 'exechdg'.

#### **4.2.19.4 'Closehdgtx;'**

This procedure returns the synchro-to-digital converter to its original condition. It also saves the heading offset to the configuration file.

'Closehdgtx' is called by 'closehdg'.

#### 4.2.20 'KEY\_CONS'

This unit provides the actual interfacing between the computer and the operator. It provides routines to read characters from the keyboard and to display messages on the console screen.

Input	:	user keyboard characters and computer messages
Output	:	characters to computer and messages on screen
Used by	:	'user'
Uses	:	'miasglob', 'dos' and 'crt'

##### 4.2.20.1 'SendMessageToDisplay( command: commandtype);'

This procedure writes the command string on the display at location (x,y) is (1,21). The rest of the line on the screen is cleared by putting spaces on these locations.

'SendMessageToDisplay' is called by 'openin\_outputdev', 'sendmessage', 'savemessage'.

##### 4.2.20.2 'Openin\_outputdev( MIASlogname: string);'

This procedure resets the variables for inputting and outputting messages. It clears the console screen, clears the message variable and opens a log file with the name in the variable 'MIASlogname'. This file is the output file to which all messages from the operator and the program are saved. Before the file is opened, a message appears on the screen to ask the operator to insert a disc if necessary and to press 'Enter'. If the log file exists, the file is opened in append mode. The available disc space and written blocks are determined also.

Note: The part where the disc number is determined is not necessary.

'Openin\_outputdev' is called by 'inituser'.

#### **4.2.20.3 'Getmessage( var command: commandtype);'**

This procedure checks if a key on the keyboard was pressed. If a key was pressed, the character is read and converted to an upcase character. If the character was a 'backspace' (code 8), the last character of the message will be deleted. If the character was not a 'backspace', the character is added at the end of the message.

If the character received was a 'carriage return' (code 13 or CR), the internal variable containing the message will be copied to the external variable 'command'. The internal variable will be cleared and the message will be send with 'sendmessage'. If the character was not a 'carriage return', the external variable will be cleared. The characters that correspond to the keys pressed on the keyboard are shown on the console screen, to provide feedback to the user about which keys were pressed.

'Getmessage' is called by 'getusermessage'.

#### **4.2.20.4 'Sendmessage( command: commandtype);'**

This procedure sends the 'command' message to 'sendmessage to display' and to 'savemessage'.

'Sendmessage' is called by 'sendusermessage'.

#### **4.2.20.5 'Sendflags( command: commandtype);'**

This procedure displays the 'command' message on the console screen at position (x,y) is (1,19). The message can be anything, but the procedure is intended to display the system flags for MIAS. This procedure was written to provide a simple but orderly console screen lay out.

'Sendflags' is called by 'senduserflags'.

#### **4.2.20.6 'Savemessage( message: commandtype);'**

This procedure saves a message to disc. It will store the message, preceded by the computer system time in seconds into the day, to provide a time tag. The 'carriage return' or 'line feed' code are deleted from the 'message'. As soon as the message is saved, the number of bits written is added to 'blockused'. If more

than 10000 blocks are written, the file is closed and opened again in append mode. The available space on the disc is updated and assigned to 'Free'. If the disc is full, a message appears on the display to ask the operator to put an empty floppy in the drive.

'Savemessage' is called by 'sendmessage' and 'saveequipmentmessage'.

#### **4.2.20.7 'Closein\_outputdev;'**

This procedure closes the log file named 'mias.dat'.

'Closein\_outputdev' is called by 'closeuser'.

#### 4.2.21 'SYNCHCNV'

This unit performs interfacing between the synchro-to-digital converter output and the units 'attbeaver' and 'hdgbeaver'. The unit was written by Dennis Willemsen and revised by Marco Meijer. The compiler is instructed to use the coprocessor or include an emulating library for the coprocessor.

Input	:	digital words representing angles
Output :		angles in Pascal variable
Used by	:	'attbeaver' and 'hdgbeaver'
Uses	:	'miasglob'

Note: 'miasglob' is used for the 'double' type definition.

##### 4.2.21.1 'Initsynchnv( var error: boolean);'

This procedure initialises the synchro-to-digital converter if necessary. It tries to find out if there is an error.

'Initsynchnv' is called by 'initattx' and 'inithdgtx'.

##### 4.2.21.2 'Shift( old:byte): byte;'

This function takes the byte 'old' and puts all bits in opposite order. Bit 1 becomes bit 8, bit 2 becomes bit 7 etc. This function was written, because the MSB from the synchro-to-digital converter is read by the computer as a LSB and vice versa.

'Shift' is called by 'Roll\_pitch\_heading'.



#### **4.2.21.3 'Roll\_pitch\_heading( nummer: integer; var hoek: double; var error: boolean);'**

This procedure instructs the synchro-to-digital converter to send the digital words that represent an angle. By giving the right instructions, one out of three synchro signals will be converted. The variable 'nummer' gives the number of the synchro which should be converted.

Because the synchro-to-digital converter is totally controlled by the address bus of the MIAS computer (PC), all instructions will be 'read' instructions. First the synchro\_to\_digital converter should be reset. This means the converter is instructed to hold the data (it cannot change the data). Then the low and high byte of the angle is read. These bytes are combined to a word and multiplied with the step size, to calculate the angle in degrees.

The angles are read twice to detect possible errors.

'Roll\_pitch\_heading' is called by 'collectatt' and 'collecthdg'.

#### **4.2.21.4 'Closesynchcnv';**

This procedure does nothing. It is here only to keep up the hierarchy and to standardize the units.

#### 4.2.22 'AR429'

This unit translates simple commands to more difficult commands for the ARINC-429-to-PC interface which is the M429PC board. This board can transmit and receive ARINC 429 signals on one channel. The board has an onboard processor and memory. This unit programs the board in the appropriate way.

The following types are declared global: 'ar429wordtype' and 'arraytype'. The 'ar429wordtype' is used to contain all bits for an ARINC 429 word. The 'arraytype' type is used to program the board to listen only to certain messages, which labels are listed in an array of 'arraytype'.

Also some constants are declared for operation in the desired mode (100 Kbit per second and a 4 bit gap between transmissions, odd parity, no SSM and no SDI). Also some variables are declared to assure smooth operation. A buffer is declared as well as pointers for this buffer and for the M429PC internal buffer.

Input	:	ARINC 429 messages as received and for transmission
Output	:	ARINC 429 messages for transmission and as received
Used by	:	'mlsbendix' and 'ar429comm'
Uses	:	'lib429p'

Note: 'ar429comm' only uses 'ar429' for the type definitions.

Note: 'lib429p' is the Pascal library as delivered with the M429PC board.

##### 4.2.22.1 'Initar429( var error: boolean; selecttable: arraytype; num: integer);'

This procedure programs the M429PC transmitter in the high speed (100 Kbit per second) mode with a gap time of 4 bit-times between the messages. The programmed information is read from the board and compared with the original information, to check for failures. Then the parity is programmed. 'No parity' is programmed, but this will be overwritten by the 'send' command if necessary.

Then the receiver is programmed for high speed operation and the 'selecttable' is loaded to the receiver. This 'selecttable' contains the labels of the ARINC 429 messages to which the M429PC board should listen. The board will be deaf for other messages. The labels in 'selecttable' should be programmed in OCTAL. This is the way the labels are used in an ARINC 429 environment. The receiver is programmed to use a circular shared buffer with 42 (maximum number of) places. Using the shared buffer, the unit can retrieve data from the buffer and put it in a bigger buffer. The last actions are to start the receiver and put the shared-buffer-pointer 'ptr1' to the first position in the shared buffer.

'Initar429' is called by 'initmlsrec'.

#### **4.2.22.2 'Getar429word( var ar429word: ar429wordtype; var noword: boolean; var a\_label: byte);'**

This procedure will read ARINC 429 words from the shared buffer and put them in the bigger MIAS computer buffer. If there is a word in the buffer, the oldest word will be read and the output-pointer will be incremented. The value of the label of the message is also determined.

Note: This label is in decimal!

Then the ARINC 429 word, which is represented in a 'longint' is converted to an array of byte and the 'noword' boolean is set to 'false'. If there was no word, the 'noword' boolean was set to 'true'.

'Getar429word' is called by 'collectmlsrec'.

#### **4.2.22.3 'Sendar429word( datain: longint; oct\_lab: byte; prate: word; var error: boolean);'**

This procedure sends an ARINC 429 word to the M429PC board. The word is composed of a label, data, SSM, SDI and a parity indicator. Then a status word should be calculated. All used slots are read, to check if the label of the message, which should be programmed, was used before. If it was used before, the same slot will be used for the new label, so there won't be two versions of one label. Then a last check is performed to see if the slot is programmed okay and the transmitter is working.

If there was no slot with the same label, the message with label is inserted in the next free slot. Then the channel is turned on and the next free slot is calculated.

'Sendar429word' is called by 'sendmlsrecommand'.

#### **4.2.22.4 'Closear429;'**

This procedure only turns off the M429PC board.

'Closear429' is called by 'closemlsrec'.

### 4.2.23 'AR429COMM'

This unit is functionally the same as 'ar429'. In the MIAS system, differential GPS information is supposed to be transmitted with the Auxiliary data words of MLS. In the MLS installation at Schiphol Airport, it is not possible (yet) to dynamically assign the Auxiliary data words. That is why the DGPS information will be transmitted using a VHF transmitter. One of the VHF-COM sets in the Beaver will be used for receiving this information.

Note: The ADW's transmitted via the VHF are coded in ARINC 429 words, because the ARINC 727 MLS receiver will output the ADW's in ARINC 429.

The VHF-COM set is connected to a MIAS computer comport using a FSK demodulator. This unit decodes the bytes received from the comport and assembles ARINC 429 words.

Input	:	bytes from comports that form ARINC 429 words, that form ADW;s
Output :		ARINC 429 words
Used by	:	'mlsbendix'
Uses	:	'ar429', 'miasglob', 'miscell' and 'com_4'

Note: 'ar429' is used for type definitions. 'miasglob' is used for a constant called 'miascfname'. 'miscell' is used to open, close, etc. the configuration file.

Note: In principle it is possible to send ARINC 429 words, but this won't be done, because it has no meaning.

Note: This unit is a preliminary version. Ewout Boks is working on a final version.

#### 4.2.23.1 'Initar429;'

This procedure will open the configuration file and read the comport number (mlsport) from the file. Then it will setup the comport, empty the associated buffers and install the interrupt handler.

Note: This procedure doesn't allow to select certain labels. Thus it is more primitive than 'initar429' from 'ar429'. This won't be a problem. The received labels are controlled by the transmitted labels, which can be controlled completely by the DGPS transmitter.

'Initar429' is called by 'initmlsrec'.

#### **4.2.23.2 'Getar429word( var ar429word: ar429wordtype; var noword: boolean; var a\_label: byte);'**

This procedure will take four bytes (an ARINC 429 word is 32 bits) from the input buffer and makes an ARINC 429 word. (If there were not enough bytes in the buffer, the procedure will be exit.) Then the parity of the ARINC 429 word is checked. If the parity was correct, the word will be output, the label will be calculated and the 'noword' boolean is set to 'false'. If the parity was incorrect and empty ARINC 429 word will be output and the 'noword' boolean is set 'true'.

'Getar429word' is called by 'collectmlsrec'.

#### **4.2.23.3 'Sendar429word;'**

This procedure is here only to make the unit complete. It does nothing.

'Sendar429word' is called by -.

#### **4.2.23.4 'Closear429;'**

This procedure removes the interrupt handler and saves the 'mlsport' variable to the configuration file.

'Closear429' is called by 'closemlsrec'.

#### 4.2.24 'ADW'

This unit contains routines for using the B/ADW card. This card reads serial bipolar return to zero data. It shifts the data in a register and interrupts the program if a preprogrammed bit pattern occurs. The unit was written by Rob Luxen.

The interface of the unit specifies compiler directives for far calls, coprocessor support and stack checking. The FIFO buffer length is specified as 1024 bytes and a time out time is specified. The FIFO is defined as an object with pointers, a buffer and functions for the FIFO. The FIFO is global to this unit, as is the timeout variable.

Only the interface of this unit will be treated here.

Input	:	bytes from the B/ADW card
Output	:	bytes in a buffer
Used by	:	'mlsbendix'
Uses	:	'intrupt', 'Crt', 'Dos'

##### 4.2.24.1 'InitKaartAdres( adres: word);'

This procedure is used to tell the unit the card address. The card uses 16 addressed.

'InitKaartAdres' is called by 'initMLSrec'.

##### 4.2.24.2 'ProgTrigFunktie( TrigNum: byte; Funktie: byte);'

This procedure programs a bit pattern in the B/ADW card. The bit pattern is formed by 'Funktie'. The function and function parity bits are used, the rest is masked. 'Trignum' indicates the number of the trigger. The triggers are numbered from 1 to 9.

'ProgTrigFunktie' is called by 'initMLSrec'.

#### **4.2.24.3 'Resettrigger( trignum: byte);'**

This procedure resets a trigger on the B/ADW board. The trigger to be reset is indicated by 'trignum'. A trigger is reset by putting a special trigger byte in the trigger register. The trigger byte is all ones except for the MSB, this will disable the comparator and so disables that trigger.

'Resettrigger' is called by 'Resettriggers'.

#### **4.2.24.4 'Resettriggers;'**

This procedure resets all triggers, by calling 'resettrigger' several times.

'Resettrigger' is called by 'install\_adw\_int'.

#### **4.2.24.5 'IRQ\_aan;'**

This procedure enables the interrupts of the B/ADW card, by setting the MSB of the first trigger.

'IRQ\_aan' is called by 'install\_adw\_int'.

#### **4.2.24.6 'IRQ\_uit;'**

This procedure disables the B/ADW card interrupt by writing a zero to the MSB of the first trigger.

'IRQ\_uit' is called by 'remove\_adw\_int'.

#### **4.2.24.7 'ResetIntLatch;'**

This procedure reset the interrupt latch on the B/ADW card by writing a byte to the reset-latch-address. It does not matter what the value of the byte is. The latch is flip flop, which has to keep the interrupt on until the interrupt routine turns it off, to prevent an interrupt to be ignored.

'Resetintlatch' is called by 'Install\_adw\_int'.

#### **4.2.24.8 'Install\_ADW\_int;'**

This procedure enables the interrupts by copying the address of the interrupt handler in the interrupt vector for interrupt 'IRQ'. 'IRQ' should not be zero, this is the system clock interrupt number. The B/ADW card contents are cleared.

'Install\_Adw\_int' is called by 'initMLSrec'.

#### **4.2.24.9 'Remove\_adw\_int;'**

This procedure restores the interrupt vector for the used interrupt and turns the B/ADW card interrupt off. It also clears the B/ADW card registers.

'Remove\_adw\_int' is called by 'closeMLSrec'.

#### **4.2.24.10 'KiesIRQ( IRQ: byte);'**

This procedure tells the unit which IRQ number is used by the B/ADW card. The IRQ number should be between 3 and 7. These are the IRQ's for comports and printers. They do not pose a threat for normal operation.

'KiesIRQ' is called by 'initMLSrec'.

#### **4.2.24.11 'FifoOBJ.ResetFIFO;'**

This procedure resets the pointers in the FIFO buffer. The interrupts should be turned off, to prevent the interrupt routine from writing in the buffer while it is being reset.

'FifoOBJ.ResetFIFO' is called by 'initMLSrec'.



#### **4.2.24.12 'FifoOBJ.FIFOempty: Boolean;'**

This function returns 'true' if the FIFO is empty. 'False' if it is not empty. The FIFO is empty is the head pointer is the same as the tail pointer.

'FifoOBJ.FIFOempty' is called by 'collectMLSrec'.

#### **4.2.24.13 'FifoOBJ.GetFIFO: byte;'**

This function returns the first byte from the FIFO. The pointer 'staart' points to this byte. 'staart' is then incremented.

'FifoOBJ.GetFIFO' is called by 'collectMLSrec'.

#### **4.2.24.14 'FifoOBJ.PutFifo( data: byte);'**

This procedure allows bytes to be put in the FIFO. The bytes are put in the buffer at the location indicated by 'kop'.

'FifoOBJ.PutFifo' is called by 'ADW\_handler' (the interrupt routine).

#### **4.2.24.15 'FifoOBJ.FIFOfull: Boolean;'**

This function returns 'true' if the FIFO is full. It returns 'false' if it is empty. The buffer is full if the head pointer is at the end of the buffer and the tail is at the begin. Or the head pointer is just one byte behind the tail pointer.

'FifoOBJ.FIFOfull' is called by 'collectMLSrec'.

## 4.2.25 'MISCELL'

This unit contains some routines which are called by many other units, but have nothing to do with positioning. These are routines for opening and closing, writing and reading from the configuration file. Furthermore there are routines for collecting the system time and some routines for adding times. Also a 'later' function is provided. Last but not least: there is a 'fileexist' function.

Global types are: 'timetype' used by 'date\_and\_time', 'addtime' and 'later'.

### 4.2.25.1 'Convert( line: string; var varname, value: string);'

This procedure analyses the input string called 'line'. It requires an input string like: #9'dgpsmode = 3;'. So the line must start with a space or a tab (#9). Then it decodes the string and assigns 'dgpsmode' to the string 'varname' and '3' to 'value'.

It is used in the initialising procedures, to initialise variables.

Syntax: #9 + sp's + varname + sp + '=' + sp + value + ';'.

or

sp's + varname + sp + '=' + sp + value + ';'.

With: #9       =       tab  
      sp       =       space  
      sp's     =       spaces

'Convert' is called by 'init', 'initgps', 'initdgps', 'initmls', 'initatt', 'inithdg', 'initgpsrec', 'initmlsrec', 'initatttx' and 'inithdgtx'.

### 4.2.25.2 'Openconfigread( var setupfile: text; filename: string);'

This procedure opens the configuration file called 'filename' and makes it ready for reading. If the file does not exist, the file is created. That is: an empty file is created containing only a 'Carriage return Line feed'. This is done, so that the program won't crash if it uses a 'readln'.

'Openconfigread' is called by 'init', 'initgps', 'initdgps', 'initmls', 'initatt', 'inithdg', 'initgpsrec', 'initmlsrec', 'initatttx' and 'inithdgtx'.

#### **4.2.25.3 'Closeconfig( var setupfile: text);'**

This procedure closes the setup file. First it checks if the file might be closed already. If not, it closes the file. If so, it doesn't do anything.

'Closeconfig' is called by 'init', 'initgps', 'initdgps', 'initmls', 'initatt', 'inithdg', 'initgpsrec', 'initmlsrec', 'initatttx' and 'inithdgtx'.

#### **4.2.25.4 'Openconfigwrite( var setupfile: text; filename: string);'**

This procedure first checks if the file called 'filename' exists. If not, a new file is created with the name 'filename'. If the file exists, the file will be opened in 'Append' mode.

'Openconfigwrite' is called by 'init', 'initgps', 'initdgps', 'initmls', 'initatt', 'inithdg', 'initgpsrec', 'initmlsrec', 'initatttx' and 'inithdgtx'.

#### **4.2.25.5 'OpenConfigWriteFirst( Var setupfile: text; filename: string);'**

This procedure opens the 'setupfile' with name 'filename' for writing. This procedure should only be called by the first procedure which writes its variables to the configuration file.

'OpenConfigWriteFirst' is called by 'Mias'.

#### **4.2.25.6 'Date\_and\_time( var time: timetype);'**

This procedure returns the current system date and time in the variable 'time'. The time is accurate to 1/100 second.

Note: The system time can be changed using the DOS commands: 'date' and 'time' or the Pascal commands: 'setdate' and 'settime'.

'Date\_and\_time' is called by 'openconfigwrite', 'collectport1', 'rawdata', 'ionoscor', 'get\_sv\_id', 'clockinfo', 'ephemeris', 'collectgpsrec', 'collectmlsrec', 'adw\_a\_conv', 'adw\_b\_conv', 'adw\_c\_conv', 'bas\_1\_conv',

'bas\_2\_conv', 'bas\_3\_conv', 'bas\_4\_conv', 'bas\_5\_conv', 'bas\_6\_conv', 'el\_conv', 'az\_conv', 'baz\_conv', 'getgpsdata', 'getdgpsdata' and 'calcpas'.

#### **4.2.25.7 'Errortime( var time: timetype);'**

This procedure fills the record 'time' with zero's, so an erroneous time will be created.

'Errortime' is called by 'GPS', 'clockinfo', 'Ephemeris', 'GPSEngine', 'init', 'MLS' and 'MLSBendix'.

#### **4.2.25.8 'Addtime( t1, t2: timetype; var sum: timetype);'**

This procedure calculates the sum of two times; t1 and t2. This includes the date. The procedure accounts for leap years. The result is put in sum.

'Addtime' is called by 'GetGPSdata', 'clockinfo', 'collectport1', 'calcpas', 'Getmlsdata' and 'collectMLSrec'.

#### **4.2.25.9 'Later( t1, t2: timetype):boolean;'**

This function returns 'true' if t1 is later than t2. It returns 'false' if t1 is not later than t2.

'Later' is called by 'GetGPSdata', 'calcpas', 'GetMLSdata' and 'collectMLSrec'.

#### **4.2.25.10 'Fileexist( filename: string): boolean;'**

This function returns 'true' if the file called 'filename' exists. It returns 'false' if it does not exist.

'Fileexist' is called by 'openin\_outputdev', 'init', 'openconfigread' and 'openconfigwrite'.

#### 4.2.26 'COM\_4'

This unit provides interfacing with the 'com' ports. This unit accesses the 8250 UART and 8259 interrupt controller directly. This is only possible on true IBM-PC compatible machines. Other register compatible chips are currently used as replacements for the 8250. They provide RS 232 or RS 422 communication facilities. The unit was written by K.R. Bulgrien and adapted by Marco Meijer.

With this unit 4 'com' ports can be used at the same time. This assumes four comports divided on two boards. The board with 'Com 3' and 'Com 4' should be changed, the 'Com 3' and 'Com 4' interrupt lines should be redirected to IRQ 5 and 7 respectively. This steals the interrupts from LPT1 and LPT2. They can't be used in interrupt mode any more. This modification of interrupt lines is only necessary if more than two comports need to be used at the same time.

The compiler is instructed to skip stack checking for the interrupts this prevents the system from crashing.

Quite a number of constants are declared globally, their function is explained in the unit.

Here only the procedures and functions that are used are discussed.

Input	:	bytes from serial line, bytes to be send
Output :		bytes to be send on serial line, bytes received
Used by	:	'gpsengine' and 'ar429comm'
Uses	:	'dos' and 'crt' .

##### 4.2.26.1 'Setupcomport( com, baud, databits, parity, stopbits: byte);'

This procedure is used to program the 8250 UART. The 'com' input can vary from 1 to 4, the baud input can from 0 to 9 representing baud rates of 110 to 38400 baud. 'databits' can vary between 5 and 8, 'parity' can be 'none', 'odd', 'null', 'even', 'markoff', 'mark', 'spaceoff' and 'space' and the number of stop bits can be 1 or 2.

'Setupcomport' is called by 'initgpsrec' and 'ar429comm.initar429'.

##### 4.2.26.2 'Inthandler1: interrupt;'

This procedure handles the interrupt from 'com 1'. This procedure is compiled in 'far' mode, so the interrupt can function properly. If a byte was received, the byte is put in the 'inbuffer' and the 'inbuffer'

pointers are adjusted. If the transmit register is empty and the transmit buffer 'outbuffer' is not empty, a byte from 'outbuffer' is put in the transmit register and the 'outbuffer' pointers are updated. Also line status changes are noted.

'Inthandler1' is called by IRQ 4.

#### **4.2.26.3 'Inthandler2: interrupt;'**

This procedure handles the interrupt from 'com 2'. This procedure is compiled in 'far' mode, so the interrupt can function properly. If a byte was received, the byte is put in the 'inbuffer' and the 'inbuffer' pointers are adjusted. If the transmit register is empty and the transmit buffer 'outbuffer' is not empty, a byte from 'outbuffer' is put in the transmit register and the 'outbuffer' pointers are updated. Also line status changes are noted.

'Inthandler2' is called by IRQ 3.

#### **4.2.26.4 'Inthandler3: interrupt;'**

This procedure handles the interrupt from 'com 3'. This procedure is compiled in 'far' mode, so the interrupt can function properly. If a byte was received, the byte is put in the 'inbuffer' and the 'inbuffer' pointers are adjusted. If the transmit register is empty and the transmit buffer 'outbuffer' is not empty, a byte from 'outbuffer' is put in the transmit register and the 'outbuffer' pointers are updated. Also line status changes are noted.

'Inthandler3' is called by IRQ 5.

#### **4.2.26.5 'Inthandler4: interrupt;'**

This procedure handles the interrupt from 'com 4'. This procedure is compiled in 'far' mode, so the interrupt can function properly. If a byte was received, the byte is put in the 'inbuffer' and the 'inbuffer' pointers are adjusted. If the transmit register is empty and the transmit buffer 'outbuffer' is not empty, a byte from 'outbuffer' is put in the transmit register and the 'outbuffer' pointers are updated. Also line status changes are noted.

'Inthandler4' is called by IRQ 7.

#### **4.2.26.6 'Installint( com: byte);'**

This procedure will put the address of the interrupt handler for 'com' port with number 'com' in the interrupt vector register for that 'com' port number. The interrupt controller is then signalled allow interrupts for this particular interrupt vector.

The old interrupt vector is stored to allow the system to return to its original state.

'Installint' is called 'initgpsrec' and 'ar429comm.ar429init'.

#### **4.2.26.7 'Removeint( com: byte);'**

This procedure will put the old interrupt vector to the interrupt vector associated with comport number 'com'.

'Removeint' is called by 'closegpsrec' and 'ar429comm.closear429'.

#### **4.2.26.8 'Emptybuffer( buffer: byte; trueinfalseout: boolean);'**

This procedure puts the pointers from 'buffer' to their zero-state. 'trueinfalseout' is used to decide which buffer should be reset. 'True' is used to reset the input buffer and 'false' is used the reset the output buffer.

'Emptybuffer' is called by 'initgpsrec' and 'ar429comm.ar429init'.

#### **4.2.26.9 'Iwritecom( com: byte; data: string);'**

This procedure is used to send a string a characters on the serial line. 'Com' indicates the 'com' port number to be used. 'Data' is the character string to be send.

The data is put in the output buffer. Then the first character is put in the transmit register of the UART. As soon as it is send, it gives and interrupt and the next character is put in the transmit register.

'Iwritecom' is called by 'execgpsrecommands' and 'ar429comm.sendar429word'.

#### **4.2.26.10 'Getcharbuff( com: byte): char;'**

This function returns the first character from the input buffer of 'com' port 'com'. If there was no character to be output, a code 0 is output.

If characters were stored in the 'restorebuff' buffer, these characters are read before the input buffer is read.

'Getcharbuff' is called by 'collectport0', 'collectport1' and 'collectmlsrec'.

#### **4.2.26.11 'Lookbuff( comport: buffer): char;'**

This function does the same as 'getcharbuff', but it does not delete the character from the buffers.

'Lookbuff' is called by 'collectport0', 'collectport1' and 'collectmlsrec'.

#### **4.2.26.12 'Charsinbuff( comport: byte): integer;'**

This function returns the number of characters in the buffer of 'com' port 'comport'. This number is calculated by adding and subtracting the in- and output pointers from the input buffer. Also the restored characters are counted.

'Charsinbuff' is called by 'collectport0', 'collectport1' and 'collectmlsrec'.

#### **4.2.26.13 'Restore\_buffer( comport: byte; line: string);'**

This procedure stores a line in a buffer. This line will be read before any other character from the input buffer. Only one line can be stored at a time. The 'restoreflag' variable indicates if something is stored in 'restorebuff'.

'Restore\_buffer' is called by 'collectport1'.



## Literature

- [1] Aeronautical Telecommunications Annex 10, Vol. 1 fourth edition.  
International Civil Aviation Organisation, April 1985.
  
- [2] Annex A to STANAG 4294 subj: NAVSTAR GPS system characteristics, Draft issue L.  
MAS NATO, 1 August 1990.
  
- [3] Leeuwen, R.G.A. van.,  
Thesis report  
Delft: Delft University of Technology, 1992.
  
- [4] MLS-20A Microwave Landing System, Maintenance manual I.B.,  
2020A Bendix General Aviation, Avionics Division.

## Index

Address_fail	-51-
Addtime	-76-
Adw_a_conv	-51-
Adw_adress	-51-
Adw_b_conv	-52-
Adw_c_conv	-53-
Adw_present	-50-
Adw_read	-27-
Arccos	-37-
Arcsin	-38-
Az_conv	-55-
Bas_1_conv	-53-
Bas_2_conv	-54-
Bas_3_conv	-54-
Bas_4_conv	-54-
Bas_5_conv	-54-
Bas_6_conv	-54-
Baz_conv	-55-
Calc_pr	-36-
Calcdgps	-27-
Calcgps	-18-
Calchybridpos	-31-
Calcmls	-21-
Calcpos	-14-
Calcsmoothpr	-36-
Charsinbuff	-80-
Clockcorrection	-33-
Clockinfo	-45-
Closear429	-67-, -69-
Closeatt	-23-
Closeatttx	-58-
Closeconfig	-75-

Closedgps	-28-
Closedown	-16-
Closegps	-19-
Closegpsrec	-47-
Closehdg	-25-
Closehdgtx	-60-
Closein_outputdev	-63-
Closemls	-21-
Closemlsrec	-56-
Closesynchcnv	-65-
Closeuser	-30-
Collectatt	-57-
Collectgpsrec	-41-
Collecthdg	-59-
Collectmlsrec	-49-
Collectport0	-41-
Collectport1	-42-
Conv_adw	-51-
Conv_ascii_2_val	-43-
Conv_bas	-53-
Conv_pos_to_wgs	-36-
Convert	-74-
Convert_pos_to_ecef	-31-
Convert_pos_to_wgs	-32-
Date_and_time	-75-
Discretes_conv	-56-
Dispflags	-13-
DW_A_coll	-52-
DW_B_coll	-52-
DW_C_coll	-53-
Eccentricanomaly	-35-
El_conv	-55-
Elev_azim	-36-
Emptybuffer	-79-
Ephemeris	-46-

Errortime	-76-
Execattcommand	-22-
Execatttxcommand	-58-
Execommands	-13-
Execdgpscommand	-27-
Execgpscommand	-18-
Execgpsrecommand	-46-
Exechdgcommand	-24-
Exechdgtxcommand	-59-
Execmlscommand	-21-
Execmlsrecommand	-56-
FifoOBJ.FIFOempty	-73-
FifoOBJ.FIFOfull	-73-
FifoOBJ.GetFIFO	-73-
FifoOBJ.ResetFIFO	-72-
Fileexist	-76-
Filterposition	-15-
Get_sv_id	-45-
Getar429word	-67-, -69-
Getattdata	-22-
Getcharbuff	-80-
Getdata	-14-
Getdgpsdata	-26-
Getgpsdata	-18-
Getgpstime	-19-
Gethdgdata	-24-
Getmessage	-62-
Getmlsdata	-20-
Getusercommands	-13-
Getusermessage	-29-
Hamming_fail	-50-
Hex2string(x:byte):string;	-56-
ifoOBJ.PutFifo	-73-
Init	-12-
Initar429	-66-, -68-

Initatt	-22-
Initatttx	-57-
Initdgps	-26-
Initgps	-17-
Initgpsrec	-40-
Initdgdg	-24-
Initdgdtx	-59-
InitKaartAdres	-70-
Initmls	-20-
Initmlsrec	-48-
Initsynchcnv	-64-
Inituser	-29-
Install_ADW_int	-72-
Installint	-79-
Inthandler1	-77-
Inthandler2	-78-
Inthandler3	-78-
Inthandler4	-78-
Inverse	-39-
Ionoscor	-45-
Ionosphericcorrection	-34-
IRQ_aan	-71-
IRQ_uit	-71-
Iwritecom	-79-
KiesIRQ	-72-
L1correction	-34-
Later	-76-
Lookbuff	-80-
Openconfigread	-74-
Openconfigwrite	-75-
OpenConfigWriteFirst	-75-
Openin_outputdev	-61-
Predictposition	-15-
ProgTrigFunktie	-70-
Rawdata	-44-

Relcorrection	-33-
Remove_adw_int	-72-
Removeint	-79-
ResetIntLatch	-71-
Resettrigger	-71-
Resettriggers	-71-
Restore_buffer	-80-
Roll_pitch_heading	-65-
Saveequipmentmessage	-30-
Savemessage	-62-
Scale	-43-
Sendar429word	-67-, -69-
Sendflags	-62-
Sendmessage	-62-
SendMessage todisplay	-61-
Sendposition	-15-
Senduserflags	-29-
Sendusermessage	-29-
SetTimetogpsifnotset	-16-
Setupcomport	-77-
Shift	-64-
Statusreport	-42-
Stopcommand	-15-
SVpos_earthadjusted	-35-
Svposition	-35-
Tan	-37-
Troposphericcorrection	-34-
Twos_complement	-27-
Twoscomplement	-44-

## Appendix A Configuration file format

The MIAS experimental airborne program uses a configuration file called 'MIAS.CFG', which contains variable names and initialising values. This is an ASCII file, which can be edited with for example the Norton Editor.

The variables are grouped per unit. This means that variable of for example the unit 'MIAS' are grouped together. The unit name precedes the variable names. The unit name is at the beginning of a new line. On the next lines, variable names can be listed preceded with a 'tab' character (ASCII value 9) or preceded by a 'space' (ASCII value 32). After the 'tab' or the 'space' comes the variable name. This is the complete name including record names and field names. The variable name is followed by one or more 'space' characters before an '=' character appears. After the '=' character, one or more 'space' characters need to be written. Then the value of the variable is written followed by a semi colon ';'.

This syntax is summarised in the following lines:

```
UNIT [cr lf]
[tab/sp]([tab/sp]) VARIABLE [sp]([sp]) = [sp]([sp]) VALUE ;[cr lf]
etc
UNIT [cr lf]
etc
```

Where [cr lf] is a carriage return line feed (ASCII characters 13 and 10), [tab/sp] is a 'tab' or 'space' character and [sp] is a 'space' character.

## Appendix B Log file format

Both the 'MIASSYST' and 'MIASLOGO' programs collect the sub system data and store it in a file called 'MIAS.DAT'. Every message from the sub system is stored preceded by the system time and a unique header. The headers and their meaning are listed in table B.1.

Table B.1 Headers for the MIAS log file.

Header	Meaning
SG	Send to the GPS receiver
RG	Received from the GPS receiver
SM	Send to the MLS receiver
RM	Received from the MLS receiver
RA	Received from the Attitude sensor
RH	Receiver from the Heading sensor

Other messages are listed in table B.2.

Table B.2 Messages in the log file.

Message	Meaning
Time set to GPS user time	The PC timer is set to indicate the GPS time as calculated by the GPS receiver. This is for time tagging
STOP	Command from the operator to stop the program
DGPSMODE = x	Change the DGPS mode. 1 is no DGPS, 2 is don't change, 3 is DGPS.

The time tag in the log file has a resolution of 1/100 of a second. This is the resolution of the PC timer. The time in the time tag is the number of seconds into the day multiplied by 100.



The sent GPS messages are stored in the file exactly the same as they were sent excluding the Carriage Return and Line Feed at the end of the message. The received GPS message are only stored if they are recognised as a valid message. This means that the 'port 0' message have to begin with '\$PMVXG' and the 'port 1' messages have to begin with 'xxx' or ' xxx', where xxx is the record number. In both cases a Line Feed has to precede the header.

The MLS angle information is stored in bits. The ARINC 429 word comes out of the ARINC 429 interface and is converted to bits, including the label and parity. Only messages with a valid parity are passed by the ARINC interface.

The attitude sensor gives one message for roll and pitch. Both have two digits after the comma and use six positions. First the roll is written, followed by a space, then the pitch is written.

The heading sensors gives a message with the heading angle using six positions and having two digits after the comma.

The B/ADW card give messages with hexadecimal numbers. Each number represents a byte received after an interrupt was given. Each hexadecimal number is separated with a space. The length of the message depends on the programmes length of a BDW or ADW. A BDW is 4 bytes and a ADW is 10 bytes. The first byte from left to right is the first byte received. The LSB of the first byte is the first bit received. The Barker code is not given.

## Appendix C Helpful programs

This appendix lists some programs which may be helpful to test the sub systems of MIAS. The program names are listed in table C.1.

Table C.1 Helpful programs and their function.

Program name	Function
COMTEST.PAS	Displays the data on comports 1 to 4 on the screen. The screen is divided in 4 parts. The baud rates are: 4800, 9600, 9600 and 9600. These can be changed. The number of bits is 8. No parity, and 1 stop bit is used.
ATT&HDG.PAS	Displays the three synchro angles as read from the synchro to digital converter.
READ_MLS.PAS	Reads the ARINC 429 angle messages of the Bendix MLS-20A receiver from a log file. It checks the status bits and converts them to a 'real', which is stored in ASCII in another file.
MLSTEST.PAS	Displays the azimuth and elevation angle as send by the Bendix MLS-20A receiver. If no angle information is received, an appropriate message is displayed.

The listings are included.

```

1 program comtest;
2
3 uses com_4, crt;
4
5 Var
6   ch   :      Char;
7   x1,
8   y1,
9   x2,
10  y2,
11  x3,
12  y3,
13  x4,
14  y4   :      Byte;
15
16 Begin
17   clrscr;
18
19   Setupcomport( 1, Ord( B4800), 8, ord( none), 1);
20   Setupcomport( 2, ord( B4800), 8, ord( none), 1);
21   Setupcomport( 3, Ord( B9600), 8, ord( none), 1);
22   Setupcomport( 4, ord( B9600), 8, ord( none), 1);
23
24   Emptybuffer( 1, True);
25   Emptybuffer( 2, True);
26   Emptybuffer( 3, True);
27   Emptybuffer( 4, True);
28
29   Installint( 1);
30   Installint( 2);
31   Installint( 3);
32   Installint( 4);
33
34   x1:= 1;
35   y1:= 1;
36   x2:= 1;
37   y2:= 1;
38   x3:= 1;
39   y3:= 1;
40   x4:= 1;
41   y4:= 1;
42
43   Repeat
44     ch:= Getcharbuff( 1);
45     If ch <> #00
46     Then Begin
47       window( 1,1, 80, 5);
48       gotoxy( x1, y1);
49       write( ch);
50       x1:= wherex;
51       y1:= wherey;
52     End;
53
54     ch:= Getcharbuff( 2);
55     If ch <> #00
56
57     Then Begin
58       window( 1,6, 80, 10);
59       gotoxy( x2, y2);
60       write( ch);
61       x2:= wherex;
62       y2:= wherey;
63     End;
64
65     ch:= Getcharbuff( 3);
66     If ch <> #00
67     Then Begin
68       window( 1,11, 80, 16);
69       gotoxy( x3, y3);
70       write( ch);
71       x3:= wherex;
72       y3:= wherey;
73     End;
74
75     ch:= Getcharbuff( 4);
76     If ch <> #00
77     Then Begin
78       window( 1,17, 80, 24);
79       gotoxy( x4, y4);
80       write( ch);
81       x4:= wherex;
82       y4:= wherey;
83     End;
84
85   Until Keypressed;
86
87   Removeint( 1);
88   Removeint( 2);
89   Removeint( 3);
90   Removeint( 4);
91 End.
92
93

```

```
1 program probeersynchro;
2
3 {$N+,E+}
4
5 Uses synchcnv, crt, miasglob;
6
7 Var
8     pitch,
9     roll,
10    heading      :      Double;
11    error        :      Boolean;
12
13 begin
14     ClrScr;
15     Writeln( 'Press any key to end');
16     initsynchcnv( error);
17     Repeat
18         gotoxy( 1,10);
19         roll_pitch_heading( 1, pitch, error);
20         roll_pitch_heading( 2, roll, error);
21         roll_pitch_heading( 3, heading, error);
22
23         writeln( 'channel 1 = ', pitch: 9: 3, ' degrees');
24         writeln( 'channel 2 = ', roll: 9:3, ' degrees');
25         writeln( 'channel 3 = ', heading: 9:3, ' degrees');
26
27     Until Keypressed;
28     closesynchcnv;
29 end.
30
```

[illegible]

```

111          { ARinc: 1,2,3 = ant no}
112
113      cnt:= 0;
114      For x:= 13 to 28 Do
115          cnt:= cnt + Ar429word[ x];
116
117      If ( cnt = 0) And ( AR429word[ 29] = 1)
118      Then Leftclr:= True
119      Else Leftclr:= False;
120
121      If ( cnt = 16) And ( Ar429word[ 29] = 0)
122      Then Rightclr:= True
123      Else Rightclr:= False;
124
125      If Leftclr Or Rightclr
126      Then Azangle_flag:= True
127      Else Begin
128          AZangle_flag:= False;
129
130      End;
131  End;
132 End;
133
134 Var
135     inputfile,
136     azfile,
137     elfile      :      Text;
138     line        :      String;
139     x           :      Byte;
140     ar429word   :      ar429wordtype;
141     code        :      Integer;
142     mlsint      :      mlsinttype;
143
144
145 Begin
146     Assign( inputfile, 'mias11.12');
147     Reset( inputfile);
148     Assign( elfile, 'el.dat');
149     Rewrite( elfile);
150     Assign( azfile, 'az.dat');
151     Rewrite( azfile);
152
153     While Not Eof( inputfile) DO
154     Begin
155         Readln( inputfile, line);
156
157         If copy( line, 13, 3) = '105'
158         Then Begin
159             If copy( line, 47, 2) = '11'
160             Then Begin
161                 for x:= 0 to 31 Do
162                 Begin
163                     Val( line[ x + 17], ar429word[x+1], code);
164                     End;
165                     Az_Conv( mlsint, ar429word);
166
167                     If mlsint.elangle_flag = false
168                     Then writeln( azfile, copy( line, 1, 7),' ', mlsint.azangle);
169                     End;
170                 End;
171             If copy( line, 13, 3) = '106'
172             Then Begin
173                 If copy( line, 47, 2) = '11'
174                 Then Begin
175                     for x:= 0 to 31 Do
176                     Begin
177                         Val( line[ x + 17], ar429word[x+1], code);
178                         End;
179                         El_Conv( mlsint, ar429word);
180                         If mlsint.azangle_flag = false
181                         Then writeln( elfile, copy( line, 1, 7),' ', mlsint.elangle);
182                         End;
183                     End;
184                 End;
185                 close( inputfile);
186                 Close( elfile);
187                 close( azfile);
188 End.
```

```

1 Program artest;
2 { this program tests Bendix MLS receiver using the Max 429 PC interface card.}
3
4 Uses AR429, crt;
5
6
7 type
8     double      =      Real;
9     flagtype    =      Boolean;
10    mlsinttype   =      Record
11
12                elangle      :      double;
13                elangle_flag :      flagtype;
14                azangle      :      double;
15                azangle_flag :      flagtype;
16                elantinue,
17                azantinue    :      byte;
18                leftclr,
19                rightclr     :      boolean;
20
21 Const
22     azimuth_lab =      151;      { label are in octal}
23     elevat_lab  =      152;      { azimuth function label}
24                                     { elevation function label}
25 Var
26     selecttable:arraytype;
27     error      :      Boolean;
28     ar429word  :      ar429wordtype;
29     noword     :      Boolean;
30     a_label    :      Byte;
31     mlsint     :      mlsinttype;
32     counter    :      Longint;
33     x          :      Integer;
34     outputfile :      Text;
35
36
37 Function Conv_BAS( ar429word: ar429wordtype; start: Byte; number: Byte)
38     : Word;
39 {*****}
40 { This function converts single bits to a number. 'Start' indicates
41   the start bit in the Arinc 429 word. 'Number' indicates the number of
42   bits to be used for the number to be formed. The number is output as a
43   Word. See Annex 10 p 60CC: LSB first.
44       Input : Arinc429 word
45             startbit
46             number of bits
47       Output: number}
48 {*****}
49 Var
50     value,
51     mult      :      Word;
52     x         :      Byte;
53
54 Begin
55     value:= 0;
56
57     mult:= 1;
58     For x:= 0 To number - 1 Do
59         Begin
60             value:= value + ar429word[ start + x] * mult;
61             mult:= mult * 2;
62         End;
63     Conv_BAS:= value;
64 End;
65
66 Procedure EL_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype);
67 {*****}
68 { This procedure converts the Arinc 429 word containing the glidepath
69   information to a Pascal variable.
70       Input : Arinc 429 word
71       Output: Mlsint}
72 {*****}
73 Begin
74     With Mlsint Do
75     Begin
76         If ( Ar429word[ 27] = 1) And
77             ( Ar429word[ 28] = 1) And
78             ( Ar429word[ 29] = 1)
79         Then Begin
80             ELangle:= Conv_bas( Ar429word, 13, 14);
81             ELangle:= ELangle - $4000;
82             ELangle:= ELangle * 0.005;
83         End;
84
85         If ( Ar429word[ 27] = 0) And
86             ( Ar429word[ 28] = 0) And
87             ( Ar429word[ 29] = 0)
88         Then ELangle:= Conv_bas( Ar429word, 13, 14) * 0.005;
89
90         ELantInUse:= Ar429word[ 12] + 1;      { bendix: 0 = aft ant}
91                                             {      1 = forward ant}
92                                             { ARinc: 1,2,3 = ant no}
93         ELangle_flag:= False;
94
95     End;
96 End;
97
98
99 Procedure AZ_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype);
100 {*****}
101 { This procedure converts the Arinc 429 word containing the azimuthangle
102   information to a Pascal variable.
103       Input : Arinc 429 word
104       Output: Mlsint}
105 {*****}
106 Var
107     x,
108     cnt      :      Byte;
109
110 Begin

```

```

111 With Mlsint Do
112 Begin
113   If ( Ar429word[ 27] = 1) And
114     ( Ar429word[ 28] = 1) And
115     ( Ar429word[ 29] = 1)
116   Then Begin
117     Azangle:= Conv_bas( Ar429word, 13, 14);
118     Azangle:= Azangle - $4000;
119     Azangle:= Azangle * 0.005;
120   End;
121
122   If ( Ar429word[ 27] = 0) And
123     ( Ar429word[ 28] = 0) And
124     ( Ar429word[ 29] = 0)
125   Then Azangle:= Conv_bas( Ar429word, 13, 14) * 0.005;
126
127   AzAntInUse:= Ar429word[ 12] + 1;      { bendix: 0 = aft ant}
128                                         {      1 = forward ant}
129                                         { ARinc: 1,2,3 = ant no}
130
131   cnt:= 0;                             { extra check possible :)
132   For x:= 13 to 28 Do                   { |angle| < propcoverage}
133     cnt:= cnt + Ar429word[ x];
134
135   If ( cnt = 0) And ( AR429word[ 29] = 1)
136   Then Leftclr:= True
137   Else Leftclr:= False;
138
139   If ( cnt = 16) And ( Ar429word[ 29] = 0)
140   Then Rightclr:= True
141   Else Rightclr:= False;
142
143   If Leftclr Or Rightclr
144   Then Azangle_flag:= True
145   Else Begin
146     Azangle_flag:= False;
147
148   End;
149 End;
150 End;
151
152
153
154
155
156 Begin
157 Assign( outputfile, 'arinc.dat');
158 Rewrite( outputfile);
159
160 ClrScr;
161 Writeln( 'Press any key to end');
162
163 selecttable[0]:= azimuth_lab;          { program the labels}
164 selecttable[1]:= elevat_lab;
165

```

```

166 InitAr429( error, selecttable, 2);
167
168 if error
169 Then Begin
170   Writeln( 'Error programming ARINC 429 receiver');
171   Exit;
172 End;
173
174 counter:= 0;
175 Repeat
176   GetAr429word( ar429word, noword, a_label);
177   If noword = false
178   Then Begin
179     For x:= 1 To 32 Do
180       Write( outputfile, ar429word[x]);
181     Writeln( outputfile);
182
183     Case a_label Of
184       105: Az_conv( mlsint, ar429word);
185       106: El_conv( mlsint, ar429word);
186     End;
187
188     Gotoxy( 1,15);
189
190     If mlsint.azangle_flag
191     Then Writeln( 'Azimuth      :          flag')
192     Else Writeln( 'Azimuth      : ', mlsint.azangle:10:5);
193
194     Gotoxy( 1, 20);
195
196     If mlsint.elangle_flag
197     Then Writeln( 'Elevation :          flag')
198     Else Writeln( 'Elevation : ', mlsint.elangle:10:5);
199
200     counter:= 10000;
201   End
202   Else Begin
203     If counter > 0
204     Then Dec( counter);
205
206     If counter = 0
207     Then Begin
208
209       Gotoxy( 1,15);
210       Writeln( 'No ARINC word received      ');
211       Gotoxy( 1,20);
212       Writeln( 'No ARINC word received      ');
213     End;
214   End;
215 Until Keypressed;
216
217 CloseAr429;
218 Close( outputfile);
219 End.

```



## **Appendix D Listings for MIASLOGO**

```

1 Unit MIASglob;
2
3 {$N+,E+}
4
5 Interface
6
7 Uses Miscell;
8
9
10 Const
11     Valid_HDGtime : timetype = (year :0;
12                                 month:0;
13                                 day :0;
14                                 hour :0;
15                                 minute:0;
16                                 sec :1;
17                                 sec100:0);
18     Valid_Atttime : timetype = (year :0;
19                                 month:0;
20                                 day :0;
21                                 hour :0;
22                                 minute:0;
23                                 sec :1;
24                                 sec100:0);
25     Valid_MLStime : timetype = (year :0;
26                                 month:0;
27                                 day :0;
28                                 hour :0;
29                                 minute:0;
30                                 sec :0;
31                                 sec100:10);
32     Valid_DGPStime : timetype = (year :0;
33                                 month:0;
34                                 day :0;
35                                 hour :0;
36                                 minute:1;
37                                 sec :30;
38                                 sec100:0);
39     Valid_GPStime : timetype = (year :0;
40                                 month:0;
41                                 day :0;
42                                 hour :0;
43                                 minute:0;
44                                 sec :1;
45                                 sec100:0);
46     MIAScfname = 'MIAS.CFG';
47
48
49 Type
50     double = Extended;
51     flagtype = Boolean; {When true, a flag is displayed}
52     integritytype= Record
53         flag: flagtype;
54     End;
55     positiontype= Record
56
57
58     WGS84lat,
59     WGS84lon,
60     WGS84alt, { in [rad] and [m]}
61     h, {altitude to MSL [km]}
62     x,
63     y,
64     z : Double;
65     flag : flagtype;
66     integrity : integritytype;
67     EcefTrueLocalFalse:Boolean;
68
69 commandtype = String;
70
71 {----- GPSdatatypes -----}
72 svpositiontype= Record {SV position}
73     x,{ECEF}
74     y,
75     z : Double;
76
77 End;
78 prntype = Record {store info per satellite}
79     pr : Double; {pseudorange}
80     rxtime : Longint;
81     intcarphase: Double; {integrated carrier}
82     flag : Boolean; {if True than error}
83     position : svpositiontype;
84
85 End;
86 GPSdatatype = Record
87     flag: flagtype; { If true then error}
88     prn : Array[1..32] Of prntype;
89     deltaT : Double; { clockerror of user clock
90                     relative to GPS clock}
91     present: Byte; { 1= present, 0=not present}
92
93 End;
94 {----- MLSdatatypes -----}
95 MLSpositiontype= Record
96     x,
97     y,
98     z : Double;
99
100 End;
101 ADWtype = Array[ 13..76] Of Byte;
102 MLSdatatype = Record
103     Azpos, { antenna positions in}
104     Elpos, { MLS reference system}
105     Bazpos, { angles are in deg}
106     DMEpos : MLSpositiontype;
107     MLSthrespos: positiontype;
108     Runwayhdg : Double;
109             { runway heading in deg}
110
111     DMErange,
112     Elangle, { corrected angles}
113     AZangle,
114     BAZangle : Double;
115     Leftclr,
116     Rightclr : Boolean;
117     ElantInUse,
118     AzantInUse: Byte;

```

```

111                                     166
112                                     167 HDGdatatype =
113                                     168
114                                     169
115                                     170
116                                     171
117                                     172 alldatatype =
118                                     173
119                                     174
120                                     175
121                                     176
122                                     177
123                                     178
124                                     179
125                                     180
126                                     181
127                                     182
128                                     183
129                                     184
130                                     185
131 {-----}
132 nablatype= Record
133     rxtime_dgps : Real;
134     order0,
135     order1      : Double;
136     flag        : flagtype;
137 End;
138
139 DGPSdatatype= Record
140     nabla      : Array [1..32] Of nablatype;
141     flag       : flagtype;
142     present: Byte; { 1= present, 0=not present}
143 End;
144 {-----}
145 LORANDatatype= Record
146     flag: flagtype;
147     present: Byte; { 1= present, 0=not present}
148 End;
149 DMEdatatype = Record
150     flag: flagtype;
151     present: Byte; { 1= present, 0=not present}
152 End;
153 Altdatatype = Record
154     flag: flagtype;
155     present: Byte; { 1= present, 0=not present}
156 End;
157 DAltdatatype= Record
158     flag: flagtype;
159     present: Byte; { 1= present, 0=not present}
160 End;
161 Attdatatype = Record
162     rollangle : Double;
163     pitchangle: Double;
164     flag       : flagtype;
165     present: Byte; { 1= present, 0=not present}
166
167 HDGdatatype =
168
169
170
171
172 alldatatype =
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192 Implementation
193
194 End.

```

```

1 Unit GPSglob;
2
3 {*****}
4 { This unit contains the types for the GPS part of MIAS.}
5 {*****}
6
7 Interface
8
9 Uses MIASglob, Miscell;
10
11 Const
12     Valid_Tck :      timetype      =      (year :0;
13                                             month:0;
14                                             day :0;
15                                             hour :5;
16                                             minute:0;
17                                             sec :0;
18                                             sec100:0);
19                                             { clock is valid for 5 hours}
20                                             { see STANAG }
21     Valid_Tephem:    timetype      =      (year :0;
22                                             month:0;
23                                             day :0;
24                                             hour :5;
25                                             minute:0;
26                                             sec :0;
27                                             sec100:0);
28                                             { ephemeris is valid for 5}
29                                             { hours, see STANAG }
30     Valid_Tionos:    timetype      =      (year :0;
31                                             month:0;
32                                             day :6;
33                                             hour :0;
34                                             minute:0;
35                                             sec :0;
36                                             sec100:0);
37                                             { ionosphere is valid for}
38                                             { 6 days}
39 Type
40     ephemeristype=    Record
41         IODE          :      Byte;
42         { Issue of data ephemeris}
43         Crs           :      Double;
44         { Amplitude of the sin harmonic
45         correction term to the orbit radius}
46         deltan        :      Double;
47         { Mean motion difference from computed value}
48         Mo            :      Double;
49         { Mean anomaly at reference time}
50         Cuc           :      Double;
51         { Amplitude of the cosine harmonic correction
52         term to the argument of latitude}
53         e             :      Double;
54         { Eccentricity}
55         Cus           :      Double;
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2
```

```

111                                     computer system
112                                     at which infor-
113                                     mation was
114                                     received.}
115                                     End;
116     ionospheretype= Record
117         alfa0      :      Double;
118         alfa1      :      Double;
119         alfa2      :      Double;
120         alfa3      :      Double;
121         beta0      :      Double;
122         beta1      :      Double;
123         beta2      :      Double;
124         beta3      :      Double;
125                                     End;
126     GPSinttype = Record
127         flag: flagtype;    { If true then error}
128         prn : Array[1..32] Of prngpstype;
129         numofsat: Shortint;{ number of sv's being
130                             tracked}
131         Deltat      :      Double; { User clock error}
132         iono: ionospheretype;
133         Tionos: timetype;
134                                     End;
135
136 Implementation
137
138 Begin
139 End.
```

```

1 Unit MLsglob;
2 {*****}
3 { This unit contains global MLStypes and MLS variables}
4 { See also the ICAO Annex 10 Part 1 page 146 and 147.}
5 {*****}
6
7 Interface
8
9 Uses Miasglob, Miscell;
10
11 Const
12     Valid_Bas1      :      timetype      =      (year :0;
13                                                    month:0;
14                                                    day  :0;
15                                                    hour :0;
16                                                    minute:0;
17                                                    sec  :1;
18                                                    sec100:0);
19     Valid_Bas2      :      timetype      =      (year :0;
20                                                    month:0;
21                                                    day  :0;
22                                                    hour :0;
23                                                    minute:0;
24                                                    sec  :0;
25                                                    sec100:16);
26     Valid_Bas3      :      timetype      =      (year :0;
27                                                    month:0;
28                                                    day  :0;
29                                                    hour :0;
30                                                    minute:0;
31                                                    sec  :1;
32                                                    sec100:0);
33     Valid_Bas4      :      timetype      =      (year :0;
34                                                    month:0;
35                                                    day  :0;
36                                                    hour :0;
37                                                    minute:0;
38                                                    sec  :1;
39                                                    sec100:0);
40     Valid_Bas5      :      timetype      =      (year :0;
41                                                    month:0;
42                                                    day  :0;
43                                                    hour :0;
44                                                    minute:0;
45                                                    sec  :1;
46                                                    sec100:0);
47     Valid_Bas6      :      timetype      =      (year :0;
48                                                    month:0;
49                                                    day  :0;
50                                                    hour :0;
51                                                    minute:0;
52                                                    sec  :1;
53                                                    sec100:0);
54     Valid_AuxA1      :      timetype      =      (year :0;
55                                                    month:0;
56                                                    day  :0;
57                                                    hour :0;
58                                                    minute:0;
59                                                    sec  :1;
60                                                    sec100:0);
61     Valid_AuxA2      :      timetype      =      (year :0;
62                                                    month:0;
63                                                    day  :0;
64                                                    hour :0;
65                                                    minute:0;
66                                                    sec  :1;
67                                                    sec100:0);
68     Valid_AuxA3      :      timetype      =      (year :0;
69                                                    month:0;
70                                                    day  :0;
71                                                    hour :0;
72                                                    minute:0;
73                                                    sec  :1;
74                                                    sec100:0);
75     Valid_AuxA4      :      timetype      =      (year :0;
76                                                    month:0;
77                                                    day  :0;
78                                                    hour :0;
79                                                    minute:0;
80                                                    sec  :1;
81                                                    sec100:0);
82     Valid_AuxB       :      timetype      =      (year :0;
83                                                    month:0;
84                                                    day  :0;
85                                                    hour :0;
86                                                    minute:2;
87                                                    sec  :0;
88                                                    sec100:0);
89     Valid_AuxC       :      timetype      =      (year :0;
90                                                    month:0;
91                                                    day  :0;
92                                                    hour :0;
93                                                    minute:0;
94                                                    sec  :3;
95                                                    sec100:0);
96     Az2thresdist= 1000;
97     AzPropCovNegLim=40;
98     AzPropCovPosLim=40;
99     Cleartype=0;
100
101     MinGP =3;
102     BAZstat=0;
103     DMEstat=0;
104     Azstat=1;
105     Elstat=1;
106
107     AzBW=1;
108     ElBW=1;
109     DMEdist=0;
110

```

```

111  AzMagOr=0;                                166                                { BackAzimuth magnetic orientation}
112  BazMagOr=0;                                167
113  BazPropCovNegLim=0;                        168
114  BazPropCovPosLim=0;                        169
115  BazBW=0;                                   170
116  {BazStat=0;}{ is already declared}         171
117                                              172
118  MLSident='AMS';                            173
119                                              174
120  AzOff =0;                                   175
121  Az2MLSdatdist =990;                        176
122  AzAlignRun=0;                              177
123  AzCoorSyst=0;                              178
124                                              179
125  Eloff=0;                                    180
126  MLSdat2thres =10;                          181
127  ElHeight =0;                               182
128                                              183
129  DMEoff=0;                                   184
130  DME2MLSdatDist=0;                          185
131                                              186
132  BAzOff=0;                                   187
133  BAz2MLSdatDist=0;                          188
134  BAzAlignRun=0;                             189
135                                              190
136 Type                                         191
137  Basic1type = Record                        192
138      Az2thresDist      : Integer;           193
139      { Azimuth to threshold distance}        194
140      AzPropCovNegLim,   195
141      { Azimuth proportional coverage,        196
142      negative limit}
143      AzPropCovPosLim,   197
144      { idem, positive limit}                 198
145      ClearType         : Byte;              199
146      { clearance signal type}                200
147                                          201
148  Basic2type = Record                        202
149      MinGP             : Real;              203
150      { minimum glide path}                   204
151      BAZstat,           205
152      { Back Azimuth status}                   206
153      DMEstat,           207
154      { DME status}                           208
155      Azstat,            209
156      { Azimuth status}                       210
157      Elstat : Byte;      { Elevation status}  211
158                                          212
159  Basic3type = Record                        213
160      AzBW,              214
161      { Azimuth beamwidth}                     215
162      ElBW,              216
163      { Elevation beamwidth}                   217
164      DMEdist : Real;   { DME distance}        218
165                                          219
166                                          220
167                                          { BackAzimuth magnetic orientation}
168                                          { Back Azimuth Proportional
169                                          Coverage Negative limit}
170                                          BazPropCovPosLim: Byte;
171                                          { idem positive limit}
172                                          BazBW : Real;
173                                          { Back Azimuth beamwidth}
174                                          BazStat : Byte;
175                                          { Back Azimuth status}
176
177                                          End;
178                                          Record
179                                          MLSident: String[3];
180                                          End;
181                                          { MLS ground equipment
182                                          identification}
183
184  AuxA1type = Record
185      AzOff, { Azimuth Antenna offset}
186      Az2MLSdatdist: Integer;
187      { Azimuth antenna to MLS datum
188      point distance}
189      AzAlignRun : Real;
190      { Azimuth Alignment with Runway
191      centreline}
192      AzCoorSyst : Byte;
193      { Azimuth Antenna Coordinate
194      system}
195
196                                          End;
197                                          Record
198      Eloff, { Elevation antenna offset}
199      MLSdat2thres: Integer;
200      { MLS datumpoint 2 threshold
201      distance}
202      ElHeight : Real;
203      { Elevation Antenna Height}
204
205                                          End;
206                                          Record
207      DMEoff, { DME offset}
208      DME2MLSdatDist: Integer;
209      { DME to MLS datum point distance}
210
211                                          End;
212                                          Record
213      BAZOff, { Back azimuth antenna offset}
214      BAZ2MLSdatDist: Integer;
215      { Back azimuth to MLS datum
216      point distance}
217      BAZAlignRun: Real;
218      { Back azimuth alignment with
219      runway centre line}
220
221                                          End;
222                                          Record
223      antenna : Byte;
224      test : Byte;

```

```

221      Azsource :      Byte;      276      Auxa1_flag,
222      Azselwarn :      Byte;      277      Auxa2_flag,
223      Bazselwarn:      Byte;      278      Auxa3_flag,
224      GPselwarn :      Byte;      279      Auxa4_flag,
225      BAZavail :      Byte;      280      AuxB_flag,
226      BAZdeven :      Byte;      281      AuxC_flag,
227      Tuningcom :      Byte;      282
228      nr1antssel :      Byte;      283      ELangle_flag,
229      changeinh :      Byte;      284      AZangle_flag,
230      tunPrtsel :      Byte;      285      BAZangle_flag,
231      End;      286      DME_flag,
232      287      discretes_flag: Boolean;
233      MLSinttype =      Record      288
234      Bas1 :      Basic1type;      289      flag : Boolean;
235      Bas2 :      Basic2type;      290
236      Bas3 :      Basic3type;      291      End;
237      Bas4 :      Basic4type;      292
238      Bas5 :      Basic5type;      293      Implementation
239      Bas6 :      Basic6type;      294
240      AuxA1 :      AuxA1type;      295      Begin
241      AuxA2 :      AuxA2type;      296      End.
242      AuxA3 :      AuxA3type;
243      AuxA4 :      AuxA4type;
244      AuxB,
245      AuxC :      ADWtype;
246
247      Bas1_Time :      timetype;
248      Bas2_Time :      timetype;
249      Bas3_Time :      timetype;
250      Bas4_Time :      timetype;
251      Bas5_Time :      timetype;
252      Bas6_Time :      timetype;
253      AuxA1_Time:      timetype;
254      AuxA2_Time:      timetype;
255      AuxA3_Time:      timetype;
256      AuxA4_Time:      timetype;
257      AuxB_time,
258      AuxC_time :      timetype;
259
260      DMErange,
261      ELangle,
262      AZangle,
263      BAZangle :      Double;
264      Discretes :      Discretetype;
265      Leftclr,
266      Rightclr :      Boolean;
267      ElantInUse,
268      AzantInUse:      Byte;
269
270      Bas1_flag,
271      Bas2_flag,
272      Bas3_flag,
273      Bas4_flag,
274      Bas5_flag,
275      Bas6_flag,

```



```
1 Program MIASLogonly;
2
3 {$M 12000, 0, 650000}
4
5 Uses MIASglob, MIAS;
6
7 Var
8   alldata      :      alldatatype;
9   command      :      commandtype;
10  position,
11  filtposition,
12  predposition :      positiontype;
13
14 Begin
15   Init( alldata, position);
16   Repeat
17     SetTimerToGPSIfNotSet;
18     GetUserCommands( command);
19     ExecCommands( command, alldata);
20     GetData( alldata);
21   Until Stopcommand( command);
22   CloseDown( alldata, position);
23 End.
```

```

1 Unit MIAS;
2
3 Interface
4
5 {$N+,E+}
6
7
8 Uses MIASglob;
9
10
11 Procedure Init( Var alldata: alldatatype; Var position: positiontype);
12 {*****}
13 {Initialise the MIAS system. Open a setupfile and read some values.
14      Input :-
15      Output:status; whether or not a device present}
16 {*****}
17
18
19 Procedure DispFlags( alldata: alldatatype; position: positiontype);
20 {*****}
21 {Display the flags on the screen or in a file. The flags represent the valid-
22 ness of a certain device. The procedure uses only the flag-fields of the
23 peripheral-fields in alldata.
24      Input :alldata
25      Output:flags on screen or file}
26 {*****}
27
28
29 Procedure GetUserCommands( Var command: commandtype);
30 {*****}
31 {Retrieve commands for the peripherals from the keyboard.
32      Input :-
33      Output:commands string in upcase}
34 {*****}
35
36
37 Procedure ExecCommands( command: commandtype; alldata: alldatatype);
38 {*****}
39 {Send commands to peripherals, or execute the commands on the host.
40      Input :commands
41      Output:-}
42 {*****}
43
44
45 Procedure GetData( Var alldata: alldatatype);
46 {*****}
47 {Retrieve information from the peripherals.
48      Input :-
49      Output:alldata; information from peripherals}
50 {*****}
51
52
53 Procedure CalcPos( Var alldata: alldatatype; Var position: positiontype);
54 {*****}
55 {Calculate the position from the information available.

```

```

56      Input :alldata
57      Output:position}
58 {*****}
59
60
61 Procedure FilterPosition( position: positiontype;
62      Var filtposition: positiontype);
63 {*****}
64 {Perform a filtering action on the positions calculated, to make sure the
65 position output will be smooth.
66      Input :position
67      Output:filtered position}
68 {*****}
69
70
71 Procedure PredictPosition( position: positiontype;
72      Var predposition: positiontype);
73 {*****}
74 {Make a prediction of the position, to counteract the calculation and
75 measurement delay.
76      Input :position
77      Output:predicted position}
78 {*****}
79
80 Procedure SendPosition( position: positiontype);
81 {*****}
82 {Send the position to another device from storage or displaying, or display
83 the position on the screen. Input :position
84      Output:position to the screen or a device}
85 {*****}
86
87
88 Function Stopcommand( command: commandtype): Boolean;
89 {*****}
90 {When a stop command is sent, this function will turn TRUE.
91      Input : commandstring
92      Output: Boolean, stop or not}
93 {*****}
94
95
96 Procedure CloseDown( alldata: alldatatype; position: positiontype);
97 {*****}
98 {Make sure the computer is back to 'normal'. Restore interruptvectors etc.
99      Input :-
100      Output:-}
101 {*****}
102
103
104 Procedure SetTimerToGPSIfNotSet;
105 {*****}
106 { Use the GPS time to set the internal timer. If the time was already set,
107 do nothing.
108      Input :-
109      Output:-}
110 {*****}

```

```

111
112
113 Implementation
114
115 Uses GPS, DGPS, MLS, Att, HDG, PosCalc, User, Miscell, crt, dos;
116
117 Const
118     Valid_tMLS      :      timetype      =      (year :0;
119                                                    month:0;
120                                                    day  :0;
121                                                    hour :0;
122                                                    minute:0;
123                                                    sec  :2;
124                                                    sec100:0);
125
126 Var
127     stopkeypressed:      Boolean;
128     DGPSflag      :      flagtype;      { Indicates DGPS mode or not}
129     DGPSmode      :      Byte;
130     allowed_error  :      Double;
131     tMLS           :      timetype;
132     Old_MLS        :      MLSDatatype;
133     timeset        :      Boolean;
134     MIASlogname    :      String;
135
136
137 Procedure Init( Var alldata: alldatatype; Var position: positiontype);
138
139 Var
140     setupfile      :      Text;
141     error           :      Boolean;
142     title,
143     varname,
144     line           :      String;
145     value          :      String;
146     code           :      Integer;
147
148 Begin
149     MIASlogname:= 'c:';
150
151     DGPSflag:= True;
152     DGPSmode:= 1;
153
154     ErrorTime( tMLS);
155     With Old_MLS Do
156     Begin
157         flag:= True;
158     End;
159
160     TimeSet:= False;
161
162     With position Do
163     Begin
164         WGS84lat:= 0;
165         WGS84lon:= 0;
166         WGS84alt:= 0;
167         h:=0;
168         x:=0;
169         y:=0;
170         z:=0;
171         flag:= True;
172         integrity.flag:= True;
173     End;
174
175     alldata.mls.MLStHresPos:= position;
176     MLSanposition:= position;
177
178     If Not FileExist( MIAScfname)
179     Then Begin
180         SendUserMessage('Configfile "MIAS.CFG" not present');
181         Halt( 1);      { terminate the program }
182     End;
183
184     OpenConfigRead( setupfile, MIAScfname);
185     Repeat      { find MIAS part of
186                                     config file}
187         Readln( setupfile, title);
188     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'MIAS'));
189
190     If Not Eof( setupfile)      { if there is more in file}
191     Then Repeat
192         Readln( setupfile, line);      { get a line}
193         Convert( line, varname, value); { extract the variable name
194                                     and value}
195         { repeat until end of file}
196         { initialise comports for
197         communication with Engine}
198
199         If ( varname = 'ALLOWED_ERROR')
200         Then Val( value, allowed_error, code);
201
202         If ( varname = 'POSITION.WGS84LAT')
203         Then Val( value, position.wgs84lat, code);
204
205         If ( varname = 'POSITION.WGS84LON')
206         Then Val( value, position.wgs84lon, code);
207
208         If ( varname = 'POSITION.WGS84ALT')
209         Then Val( value, position.wgs84alt, code);
210
211         If ( varname = 'DGPSMODE')
212         Then Val( value, dgpsmode, code);
213
214         If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84LAT')
215         Then Val( value, alldata.mls.mlsthrespos.wgs84lat, code);
216
217         If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84LON')
218         Then Val( value, alldata.mls.mlsthrespos.wgs84lon, code);
219
220         If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84ALT')

```

```

221      Then Val( value, alldata.mls.mlsthrespos.wgs84alt, code);
222
223      If ( varname = 'ALLDATA.POS_ZEROVECTOR.X')
224      Then Val( value, alldata.pos_zerovector.x, code);
225
226      If ( varname = 'ALLDATA.POS_ZEROVECTOR.Y')
227      Then Val( value, alldata.pos_zerovector.y, code);
228
229      If ( varname = 'ALLDATA.POS_ZEROVECTOR.Z')
230      Then Val( value, alldata.pos_zerovector.z, code);
231
232      If ( varname = 'ALLDATA.ANT_ZEROVECTOR.X')
233      Then Val( value, alldata.ant_zerovector.x, code);
234
235      If ( varname = 'ALLDATA.ANT_ZEROVECTOR.Y')
236      Then Val( value, alldata.ant_zerovector.y, code);
237
238      If ( varname = 'ALLDATA.ANT_ZEROVECTOR.Z')
239      Then Val( value, alldata.ant_zerovector.z, code);
240
241      If ( varname = 'ALLDATA.GPS.PRESENT')
242      Then Val( value, alldata.gps.present, code);
243
244      If ( varname = 'ALLDATA.DGPS.PRESENT')
245      Then Val( value, alldata.dgps.present, code);
246
247      If ( varname = 'ALLDATA.MLS.PRESENT')
248      Then Val( value, alldata.mls.present, code);
249
250      If ( varname = 'ALLDATA.ATT.PRESENT')
251      Then Val( value, alldata.att.present, code);
252
253      If ( varname = 'ALLDATA.HDG.PRESENT')
254      Then Val( value, alldata.hdg.present, code);
255
256      If ( varname = 'MIASLOGNAME')
257      Then MIASlogname:= value;
258      Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
259      CloseConfig( setupfile);
260
261      Convert_Pos_to_Ecef( position);
262      Convert_Pos_to_Ecef( alldata.mls.mlsthrespos);
263
264      MLSanposition:= position;
265
266      InitUser( MIASlogname);
267
268      With alldata Do
269      Begin
270          InitGPS( gps);
271          InitMLS( mls);
272          InitAtt( att);
273          InitHDG( hdg);
274      End;
275

```

```

276 End;
277
278
279 Procedure DispFlags( alldata: alldatatype; position: positiontype);
280
281 Var
282     line          :      Commandtype;
283
284 Begin
285     line:= '';
286     With alldata Do
287     Begin
288         If Not GPS.flag Then line:= line + 'GPS oke '
289         Else line:= line + 'GPS err ';
290
291         If Not DGPS.flag Then line:= line + 'DGPS oke '
292         Else line:= line + 'DGPS err ';
293
294         If Not MLS.flag Then line:= line + 'MLS oke '
295         Else line:= line + 'MLS err ';
296
297         If Not Att.flag Then line:= line + 'Att oke '
298         Else line:= line + 'Att err ';
299
300         If Not HDG.flag Then line:= line + 'HDG oke '
301         Else line:= line + 'HDG err ';
302
303         If Not position.flag Then line:= line + 'Pos oke '
304         Else line:= line + 'Pos err ';
305
306         If ( position.integrity.flag) Then line:= line + 'int err '
307         Else line:= line + 'int oke ';
308     SendUserFlags( line);
309     End;
310 End;
311
312
313 Procedure GetUserCommands( Var command: commandtype);
314
315 Begin
316     GetUserMessage( command);
317 End;
318
319
320 Procedure ExecCommands( command: commandtype; alldata: alldatatype);
321
322 Var
323     substr          :      String;
324     varname,
325     value           :      String;
326     code            :      Integer;
327     tempdgpsmode   :      Byte;
328
329 Begin
330     substr:= Copy( command, 1, 4);

```

```

331
332   If ( substr = 'GPS:') And ( alldata.gps.present = 1)
333   Then ExecGPScommand( command);
334
335   If ( substr = 'MLS:') And ( alldata.mls.present = 1)
336   Then ExecMLScommand( command);
337
338   If ( substr = 'ATT:') And ( alldata.att.present = 1)
339   Then ExecAttcommand( command);
340
341   If ( substr = 'HDG:') And ( alldata.hdg.present = 1)
342   Then ExecHDGcommand( command);
343
344   substr:= Copy( command, 1, 5);
345
346   If ( substr = 'DGPS:') And ( alldata.dgps.present = 1)
347   Then ExecDGPScommand( command);
348
349   If substr = 'MIAS:'
350   Then Begin
351       substr:= Copy( command, 6, Length( command) - 5);
352       Convert( ' '+substr, varname, value);{ add space for}
353                                           { procedure convert}
354       If (varname = 'DGPSMODE')
355       Then Begin
356           Val( value, tempdgpsmode, code);
357           If code <> 0
358           Then SendUserMessage( 'Error')
359           Else dgpsmode:= tempdgpsmode;
360       End;
361   End;
362 End;
363
364 Procedure GetData( Var alldata: alldatatype);
365
366 Begin
367     With alldata Do
368     Begin
369         If ( gps.present = 1)
370         Then GetGPSdata( GPS, DGPSflag);
371
372         If ( mls.present = 1) Or           { call getmlsdata also}
373         ( dgps.present = 1)             { for dgps data}
374         Then GetMLSdata( MLS);
375
376         If ( dgps.present = 1)
377         Then GetDGPSdata( MLS, DGPS);
378
379         If Not gps.flag
380         Then Begin
381             If ( att.present = 1)
382             Then GetAttdata( Att);
383
384             If ( hdg.present = 1 )

```

```

386         Then GetHDGdata( HDG);
387     End;
388 End;
389 End;
390
391 Procedure CalcPos( Var alldata: alldatatype; Var position: positiontype);
392
393 Const
394     MaxNumOfIt =      1;                { 1 is for no iterations}
395
396 Var
397     old_deltaT      :      Double;
398     iter             :      Integer;
399     t,
400     result           :      timetype;
401
402 Begin
403     Case DGPSmode Of
404     1: DGPSflag:= False;                { no Diff GPS}
405     2: ;                                { only DGPS in prop cov}
406     3: DGPSflag:= True;                 { always DGPS if available}
407
408     End;
409
410     If ( alldata.mls.present = 1)
411     Then Begin
412         CalcMLS( alldata.MLS);
413
414         If Not alldata.mls.flag          { if valid info, then adjust}
415         Then Date_and_time( tMLS);      { receive time}
416
417         Date_and_time( t);               { get current system time}
418         AddTime( tMLS, Valid_tMLS, result);
419         { MLS only valid for 'Valid_
420         tMLS' seconds}
421         If alldata.MLS.flag And          { if no valid info and}
422         Not Later( t, result)            { no time out}
423         Then alldata.MLS:= Old_MLS      { use old info}
424         Else Old_MLS:= alldata.MLS;     { store valid info for later
425                                         use}
426     End;
427
428     If ( alldata.gps.present = 1)
429     Then CalcGPS( alldata.GPS, position, DGPSflag);
430
431     If ( alldata.dgps.present = 1)
432     Then CalcDGPS( alldata);
433
434     CalchybridPos( alldata, allowed_error, position);
435 End;
436
437 Procedure FilterPosition( position: positiontype;
438                         Var filtposition: positiontype);
439
440

```

```

441 Begin
442 End;
443
444
445 Procedure PredictPosition( position: positiontype;
446                           Var predposition: positiontype);
447 Begin
448 End;
449
450
451 Procedure SendPosition( position: positiontype);
452
453 Var
454   line      : String;
455   dum       : String;
456
457 Begin
458   If position.flag
459   Then Exit;
460
461   line:= '';
462   With position DO
463   Begin
464     If EcefTrueLocalFalse
465     Then Begin
466       Str( wgs84lat * 180 / pi:23, dum);
467       line:= line + 'lat = ' + dum + ' ';
468       Str( wgs84lon * 180 / pi:23, dum);
469       line:= line + 'lon = ' + dum + ' ';
470       Str( wgs84alt:23, dum);
471       line:= line + 'alt = ' + dum + ' ';
472       SendUserMessage( line);
473
474       Str( x:23, dum);
475       line:= line + 'x = ' + dum + ' ';
476       Str( y:23, dum);
477       line:= line + 'y = ' + dum + ' ';
478       Str( z:23, dum);
479       line:= line + 'z = ' + dum + ' ';
480       SendUserMessage( line);
481     End
482   Else Begin
483     Str( x:23, dum);
484     line:= line + 'a = ' + dum + ' ';
485     Str( y:23, dum);
486     line:= line + 'y = ' + dum + ' ';
487     Str( z:23, dum);
488     line:= line + 'z = ' + dum + ' ';
489     SendUserMessage( line);
490   End;
491 End;
492 End;
493
494
495 Function Stopcommand( command: commandtype): Boolean;

```

```

496
497 Begin
498   If Copy( command, 1, 4) = 'STOP'
499   Then stopcommand:= True
500   Else stopcommand:= False;
501 End;
502
503
504 Procedure CloseDown( alldata: alldatatype; position: positiontype);
505
506 Var
507   setupfile      : Text;
508   Value          : String;
509
510 Begin
511   OpenConfigWriteFirst( setupfile, MIAScfname);
512   Writeln( setupfile, 'MIAS');
513
514   Str( allowed_error, value);
515   Writeln( setupfile, #9'allowed_error = ', value, ';');
516
517   With position Do
518   Begin
519     Str( WGS84lat, value);
520     Writeln( setupfile, #9'position.wgs84lat = ', value, ';');
521
522     Str( WGS84lon, value);
523     Writeln( setupfile, #9'position.wgs84lon = ', value, ';');
524
525     Str( WGS84alt, value);
526     Writeln( setupfile, #9'position.wgs84alt = ', value, ';');
527   End;
528   Str( dgpsmode, value);
529   Writeln( setupfile, #9'dgpsmode = ', value, ';');
530
531   With alldata.mls.MLsthrespos Do
532   Begin
533     Str( wgs84lat, value);
534     Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84lat = ',
535             value, ';');
536
537     Str( wgs84lon, value);
538     Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84lon = ',
539             value, ';');
540
541     Str( wgs84alt, value);
542     Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84alt = ',
543             value, ';');
544   End;
545
546   With alldata.Pos_zerovector Do
547   Begin
548     Str( x, value);
549     Writeln( setupfile, #9'alldata.pos_zerovector.x = ', value, ';');
550

```

```

551      Str( y, value);
552      Writeln( setupfile, #9'alldata.pos_zerovector.y = ', value, ');');
553
554      Str( z, value);
555      Writeln( setupfile, #9'alldata.pos_zerovector.z = ', value, ');');
556 End;
557
558 With alldata.Ant_zerovector Do
559 Begin
560      Str( x, value);
561      Writeln( setupfile, #9'alldata.ant_zerovector.x = ', value, ');');
562
563      Str( y, value);
564      Writeln( setupfile, #9'alldata.ant_zerovector.y = ', value, ');');
565
566      Str( z, value);
567      Writeln( setupfile, #9'alldata.ant_zerovector.z = ', value, ');');
568 End;
569
570 With alldata Do
571 Begin
572      Str( gps.present, value);
573      Writeln( setupfile, #9'alldata.gps.present = ', value, ');');
574
575      Str( dgps.present, value);
576      Writeln( setupfile, #9'alldata.dgps.present = ', value, ');');
577
578      Str( mls.present, value);
579      Writeln( setupfile, #9'alldata.mls.present = ', value, ');');
580
581      Str( att.present, value);
582      Writeln( setupfile, #9'alldata.att.present = ', value, ');');
583
584      Str( hdg.present, value);
585      Writeln( setupfile, #9'alldata.hdg.present = ', value, ');');
586 End;
587
588 Writeln( setupfile, #9'miaslogname = ', miaslogname, ');');
589
590 CloseConfig( setupfile);
591
592 With alldata Do
593 Begin
594      If ( gps.present = 1)
595      Then CloseGPS( gps, position);
596
597      If ( dgps.present = 1)
598      Then CloseDGPS( dgps);
599
600      If ( mls.present = 1)
601      Then CloseMLS( mls);
602
603      If ( att.present = 1)
604      Then CloseAtt( att);
605

```

```

606      If ( hdg.present = 1)
607      Then CloseHDG( hdg);
608 End;
609
610 CloseUser;
611 End;
612
613
614
615 Procedure SetTimerToGPSIfNotSet;
616
617 Var
618     gpstime      :      Longint;
619     valid         :      boolean;
620     hour,
621     minute,
622     sec,
623     sec100,
624     year,
625     month,
626     day,
627     dayofweek    :      Word;
628
629
630 Begin
631     If Not TimeSet
632     Then Begin
633         GetGPSTime( gpstime, valid);
634         If valid
635         Then Begin
636             day:= gpstime div 86400;
637             gpstime:= gpstime Mod 86400;
638
639             hour:= gpstime div 3600;
640             gpstime:= gpstime Mod 3600;
641
642             minute:= gpstime div 60;
643             gpstime:= gpstime Mod 60;
644
645             sec:= gpstime;
646             sec100:= 0;
647
648             SetTime( hour, minute, sec, sec100);
649             TimeSet:= True;
650
651             SendUserMessage(
652                 'Time set to User time of GPS receiver');
653         end;
654     End;
655 End;
656
657
658 Begin
659 End.

```

```

1 Unit GPS;
2
3 Interface
4
5 {$N+,E+}
6
7 Uses MIASglob, crt;
8
9 Procedure InitGps( Var GPSdata: gpsdatatype);
10 {*****}
11 {Initialise the GPS part of the system.
12   Input :-
13   Output:error; If something went wrong, error is
14   set to True}
15 {*****}
16
17
18 Procedure GetGPSdata( Var GPSdata: GPSdatatype; DGPS: Boolean);
19 {*****}
20 {Retrieve data from the GPS receiver connected to the system. Check the col-
21 lected data for age. When True, the DGPS flag indicates that DGPS mode is
22 active, and only ephemeris and pseudoranges should be valid.
23   Input :DGPS flag
24   Output:GPSdata; relevant data from the GPS
25   receiver}
26 {*****}
27
28
29 Procedure CalcGPS( Var GPSdata: GPSdatatype; position: positiontype;
30   DGPS: Boolean);
31 {*****}
32 { This procedure executes the necessary calculations for GPS. If the DGPS
33 flag is true, only the SV position and the Elevation and Azimuth to the SV
34 are calculated (For DGPS no corrections for clock, ionosphere, troposphere
35 etc are needed). If the DGPS flag is false, then all these corrections are
36 needed.
37   Input : GPSdata, ephemeris etc
38   DGPS flag
39   Output: GPSdata, SV positions}
40 {*****}
41
42
43 Procedure ExecGPScommand( command: commandtype);
44 {*****}
45 { This procedure receives a command destined for the GPS part of the MIAS
46 system. It passes the command on to the GPSreceiver-part of the system.
47   Input : command
48   Output:-}
49 {*****}
50
51
52 Procedure CloseGps( GPSdata: GPSdatatype; position: positiontype);
53 {*****}
54 {Make sure the GPS part is back to 'normal'.
55   Input :-
56
57   Output:-}
58 {*****}
59
60 Procedure GetGPStime( Var gpstime: Longint; Var valid: boolean);
61 {*****}
62 { Get the GPStime from the 'gpsint' variable. The resolution is in seconds.
63   Input :-
64   Output:GPStime and a valid boolean}
65 {*****}
66
67
68 Implementation
69
70 Uses GPSEngine, GPSCalc, GPSglob, Miscell, user;
71
72 Var
73   GPSint      :      GPSinttype;
74   El_limit    :      Byte;           { satellite elevation limit}
75   HorAccFac   :      Real;          { horizontal acceleration
76                                     factor}
77   x           :      Integer;       {counter for initialising}
78
79
80 Procedure InitGps( Var GPSdata: gpsdatatype);
81
82 Var
83   setupfile   :      Text;
84   title       :      String;
85   line        :      String;
86   varname     :      String;
87   value       :      String;
88   code        :      Integer;
89   deg         :
90   min         :      Double;
91   time        :      timetype;
92   position    :      positiontype;
93   x           :      Integer;       {counter for initialising}
94   error       :      Boolean;
95
96 Begin
97   error:= True;
98   GPSdata.flag:= error;
99   GPSdata.deltaT:= 0;
100  For x:= 1 To 32 Do
101    With GPSdata. prn[ x] Do
102      Begin
103        flag:= True;
104        pr := 0;
105      End;
106  If ( gpsdata.present = 0)
107  Then Exit;
108
109  OpenConfigRead( setupfile, MIAScfname);
110  Repeat

```

{ find GPS part of



```

111          config file;
112      Readln( setupfile, title);
113      Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'GPS')));
114
115      If Not Eof( setupfile)          { if there is more in file}
116      Then Repeat
117          Readln( setupfile, line);    { get a line}
118          Convert( line, varname, value); { extract the variable name
119                                          and value}
120                                          { repeat until end of file}
121          If ( varname = 'POSITION.WGS84LAT')
122          Then Val( value, position.wgs84lat, code);
123
124          If ( varname = 'POSITION.WGS84LON')
125          Then Val( value, position.wgs84lon, code);
126
127          If ( varname = 'POSITION.WGS84ALT')
128          Then Val( value, position.wgs84alt, code);
129
130          If ( varname = 'EL_LIMIT')
131          Then Val( value, el_limit, code);
132
133          If ( varname = 'HORACCFAC')
134          Then Val( value, horaccfac, code);
135      Until ( Eof( setupfile) Or ( Pos( ' ', line) = 0));
136      CloseConfig( setupfile);
137
138      InitGPSrec( error);
139
140      ExecGPSrecommand( 'GPS:RESET');
141
142      Date_and_time( time);
143      With time Do
144      Begin
145          Str( day, value);
146          value:= Copy( zeros, 1, 2 - Length( value)) + value;
147          line:= value + ' ';
148          { It contains the date and}
149          { time, the latitude, longi-}
150          { tude and altitude of the}
151          Str( month, value);
152          value:= Copy( zeros, 1, 2 - Length( value)) + value;
153          line:= line + value + ' ';
154          { last position}
155          Str( year, value);
156          value:= Copy( value, 3, 2);
157          { don't take the 19 from }
158          { 19xx, only take xx}
159          value:= Copy( zeros, 1, 2 - Length( value)) + value;
160          line:= line + value + ' ';
161          { add zeros}
162          Str( hour, value);
163          value:= Copy( zeros, 1, 2 - Length( value)) + value;
164          line:= line + value + '00 ';
165          { add zeros}
166      End;

```

```

166      With position Do
167      Begin
168          deg:= Abs( Trunc( wgs84lat * 180 / pi));
169          Str( deg :2 : 0, value);
170          While value[ 1] = ' ' Do          { delete leading spaces}
171              value:= Copy( value, 2, Length( value) -1);
172          value:= Copy( zeros, 1, 2 - Length( value)) + value;
173          line:= line + value;              { add zeros}
174
175          min:= 60 * ( Abs( wgs84lat * 180 / pi) - deg);
176          Str( min :7 : 4, value);
177          While value[ 1] = ' ' Do          { delete leading spaces}
178              value:= Copy( value, 2, Length( value) -1);
179          value:= Copy( zeros, 1, 7 - Length( value)) + value;
180          line:= line + value + ' ';        { add zeros}
181
182          If deg < 0
183          Then line:= line + 'S '
184          Else line:= line + 'N ';
185
186          deg:= Abs( Trunc( wgs84lon * 180 / pi));
187          Str( deg :2 :0, value);
188          While value[ 1] = ' ' Do          { delete leading spaces}
189              value:= Copy( value, 2, Length( value) -1);
190          value:= Copy( zeros, 1, 3 - Length( value)) + value;
191          line:= line + value;              { add zeros}
192
193          min:= 60 * ( Abs( wgs84lon * 180 / pi) - deg);
194          Str( min :7 :4, value);
195          While value[ 1] = ' ' Do          { delete leading spaces}
196              value:= Copy( value, 2, Length( value) -1);
197          value:= Copy( zeros, 1, 7 - Length( value)) + value;
198          line:= line + value + ' ';        { add zeros}
199
200          If deg > 0
201          Then line:= line + 'E '
202          Else line:= line + 'W ';
203
204          If Abs( wgs84alt) > 99999.9          { if overflow}
205          Then value:= '99999.9'              { then take maximum value}
206          Else Str( wgs84alt :7 :1, value);
207          While value[ 1] = ' ' Do          { delete leading spaces}
208              value:= Copy( value, 2, Length( value) -1);
209          If wgs84alt < 0
210          Then value:= '-' + Copy( zeros, 1, 7 - Length( value)) +
211                      Copy( value, 2, Length( value))
212          Else value:= Copy( zeros, 1, 7 - Length( value)) + value;
213          line:= line + value;              { add zeros}
214      End;
215      Str( HorAccFac :4 :1, value);
216      While value[1] = ' ' Do
217          value:= Copy( value, 2, Length( value) -1);
218      value:= Copy( zeros, 1, 4 - Length( value)) + value;
219      line:= line + ' ' + value;
220

```

```

221   Str( el_limit: 2, value);
222   While value[1] = ' ' Do
223     value:= Copy( value, 2, Length( value) -1);
224   value:= Copy( zeros, 1, 2 - Length( value)) + value;
225   line:= line + ' ' + value;
226
227   ExecGPSrecommand( 'GPS:INIT '+ line);
228   ExecGPSrecommand( 'GPS:SEND EPHEMERIS ETC');
229
230   GPSdata.flag:= error;
231 End;
232
233
234 Procedure GetGPSdata( Var GPSdata: GPSdatatype; DGPS: Boolean);
235
236
237 Var
238   currenttime,
239   result      :    timetype;
240   x           :    Byte;
241
242 Begin
243   CollectGPSrec( GPSint);
244
245   Date_and_Time( currenttime);           { check if data valid}
246
247   GPSint.flag:= True;                    { begin with assumption}
248                                           { that info is incorrect}
249   For x:= 1 To 32 Do
250     GPSint.flag:= GPSint.flag And GPSint.prn[x].flag;
251
252   For x:= 1 To 32 Do                    { update external var}
253     GPSdata.prn[x].flag:= GPSint.prn[x].flag;
254   GPSdata.flag:= GPSint.flag;
255 End;
256
257
258 Procedure CalcGPS( Var GPSdata: GPSdatatype; position: positiontype;
259                   DGPS: Boolean);
260
261 Var
262   sv_id       :    Byte;
263   line,
264   dum         :    String;
265
266 Begin
267   If GPSint.flag           { GPSint not valid}
268   Then Exit;               { GPSint is a global
269                           variable in the GPS units}
270
271   With GPSint Do
272     For sv_id:= 1 To 32 Do
273       Begin
274         If Not prn[sv_id].flag
275           Then Begin
276             gotoxy (1,5);
277             write (prn[sv_id].rxtime:10:2);
278             GPSdata.prn[sv_id]. rxtime:= Round(GPSint.prn[sv_id].rxtime);
279
280             { note: in poscalc deltaT}
281             { is subtracted. Doing so}
282             { here would be obsolete}
283             Clockcorrection( sv_id, GPSint); { always correct for clock}
284             RelCorrection( sv_id, GPSint);
285             If ( Not DGPS) And ( Not position.flag)
286             Then Begin
287               L1correction( sv_id, GPSint);
288               Convert_Pos_to_WGS( position);
289               Elev_Azim( sv_id, GPSint, position);
290               Ionosphericcorrection( sv_id, position, GPSint);
291               Troposphericcorrection( sv_id, position, GPSint);
292             End;
293
294             SVposition( sv_id, GPSint);
295             Calc_Pr( sv_id, GPSint);
296             gpsint.deltat:= gpsdata.deltat; { deltaT from position
297                                           calculation copied to
298                                           internal variable to
299                                           be used to correct
300                                           satellite position}
301             SVpos_earthadjusted( sv_id, GPSint);
302
303             GPSdata.prn[sv_id].position:= GPSint.prn[sv_id].position;
304             GPSdata.prn[sv_id].pr:= GPSint.prn[sv_id].pr;
305             GPSdata.prn[sv_id].intcarphase:=GPSint.prn[sv_id].intcarphase;
306           End;
307         End;
308       End;
309
310
311 Procedure ExecGPScommand( command: commandtype);
312
313 Begin
314   ExecGPSrecommand( command);
315 End;
316
317
318 Procedure CloseGps( GPSdata: GPSdatatype; position: positiontype);
319
320 Var
321   x      :    Byte;
322   setupfile :    Text;
323   value   :    String;
324
325
326 Begin
327   OpenConfigWrite( setupfile, MIAScfgname);
328   Writeln( setupfile, 'GPS');
329
330   With position Do

```

```

331 Begin
332   Str( WGS84lat, value);
333   Writeln( setupfile, #9'position.wgs84lat = ', value, ');');
334
335   Str( WGS84lon, value);
336   Writeln( setupfile, #9'position.wgs84lon = ', value, ');');
337
338   Str( WGS84alt, value);
339   Writeln( setupfile, #9'position.wgs84alt = ', value, ');');
340 End;
341
342 Str( el_limit, value);
343 Writeln( setupfile, #9'el_limit', ' = ', el_limit, ');');
344
345 Str( horaccfac, value);
346 Writeln( setupfile, #9'horaccfac', ' = ', horaccfac, ');');
347
348 CloseConfig( setupfile);
349
350 CloseGPSrec;
351 End;
352
353
354 Procedure GetGPStime( Var gpstime: Longint; Var valid: boolean);
355
356 Var
357   x      :      Byte;
358
359 Begin
360   If gpsint.flag = true
361   Then Begin
362     valid:= False;
363     Exit;
364   End
365   Else Begin
366     x:=1;
367     While ( gpsint.prn[x].flag = true) And
368           ( x < 32) Do
369       Inc( x);
370     If x <= 32
371     Then Begin
372       gpstime:= Round( gpsint.prn[x].rxtime);
373       valid:= True;
374     End
375     Else valid:= False;
376   End;
377 End;
378
379
380 Begin                                     { initialising part}
381   GPSint.flag:= True;
382   GPSint.numofsat := 0;
383   GPSint.deltaT:= 0;
384   ErrorTime( GPSint.Tionos);
385   For x:= 1 To 32 Do
386     With GPSint.prn[x] Do
387     Begin
388       flag:= True;
389       Ek:=0;
390       pr:= 0;
391       ErrorTime( Tck);
392       ErrorTime( Tephem);
393     End;
394   EL_limit:= 0;
395   HorAccFac:= 0;
396
397 End.
```

```

1 Unit MLS;
2
3 {$N+,E+}
4
5 Interface
6
7 Uses MIAsglob;
8
9 Procedure InitMLS( Var MLSdata: MLSdatatype);
10 {*****}
11 {Initialise the MLS sensor. If something went wrong, the error flag is set
12  to the value True.      Input :-
13                           Output:error}
14 {*****}
15
16
17 Procedure GetMLSdata( Var MLSdata: MLSdatatype);
18 {*****}
19 {Retrieve MLS data from the MLS sensor.
20  Input :-
21  Output:MLSdata}
22 {*****}
23
24
25
26 Procedure CalcMLS( Var MLSdata: MLSdatatype);
27 {*****}
28 {This procedure performs the necessary MLS calculations.
29  Input : MLSdata
30  Output: MLSdata.}
31 {*****}
32
33
34 Procedure ExecMLScommand( command: commandtype);
35 {*****}
36 {This procedure sends commands to the MLS receiver.
37  Input : command
38  Output: -}
39 {*****}
40
41
42 Procedure CloseMLS( MLSdata: MLSdatatype);
43 {*****}
44 {Closedown the MLS sensor. Input :-
45  Output:-}
46 {*****}
47
48
49
50 Implementation
51
52
53 Uses MLSbendix, MLSglob, Miscell, crt, user;
54
55 Var

```

```

56 MLSint      :      MLSinttype;
57
58
59 Procedure InitMLS( Var MLSdata: MLSdatatype);
60
61 Var
62  setupfile   :      Text;
63  title       :      String;
64  line        :      String;
65  varname     :      String;
66  value       :      String;
67  code        :      Integer;
68  error       :      Boolean;
69
70 Begin
71  error:= True;
72  MLSdata.flag:= error;
73
74  If ( mlsdata.present = 0)
75  Then Exit;
76
77  OpenConfigRead( setupfile, MIAScfgname);
78  Repeat
79
80      Readln( setupfile, title);
81  Until (EOF( setupfile) OR ( Copy( title, 1, 3) = 'MLS'));
82
83  If Not Eof( setupfile)
84  Then Repeat
85      Readln( setupfile, line);
86      Convert( line, varname, value);
87
88      { extract the variable name
89      and value}
90      { repeat until end of file}
91      { initialise comports for
92      communication with Engine}
93  Until ( Eof( setupfile) Or ( Pos( ' ', line) = 0));
94  CloseConfig( setupfile);
95
96  InitMLSrec( error);
97  MLSdata.flag:= error;
98 End;
99
100 Procedure GetMLSdata( Var MLSdata: MLSdatatype);
101
102 Var
103  result,
104  currenttime :      Timetype;
105
106 Begin
107  CollectMLSrec( MLSint);
108 End;
109
110

```

```

111 Procedure CalcMLS( Var MLSdata: MLSdatatype);
112
113 Begin
114   With MLSdata Do
115     Begin
116       { calculate the el, az and}
117       { baz position referenced}
118       { to the MLS datum point}
119       { MLSint is a global var-}
120       { iable in the MLS units}
121       x:= -1 * MLSint.AuxA1.Az2MLSdatdist;
122       y:= MLSint.AuxA1.Azoff;
123       z:= 0;
124       Azpos_flag:= False;
125     End;
126   With Elpos Do
127     If Not MLSint.AuxA2_flag
128     Then Begin
129       x:= 0;
130       y:= MLSint.AuxA2.Elloff;
131       z:= MLSint.AuxA2.ElHeight;
132       Elpos_flag:= False;
133     End;
134   With Bazpos Do
135     If ( Not MLSint.AuxA4_flag)
136     Then Begin
137       x:= MLSint.AuxA4.Baz2MLSdatdist;
138       y:= MLSint.AuxA4.Bazoff;
139       z:= 0;
140       Bazpos_flag:= False;
141     End;
142   With DMEpos Do
143     If ( Not MLSint.AuxA3_flag)
144     Then Begin
145       x:= -1 * MLSint.AuxA3.DME2MLSdatdist;
146       y:= MLSint.AuxA3.DMEoff;
147       z:= 0;
148       DMEpos_flag:= False;
149     End;
150   If (Not MLSint.ELangle_flag) And (MLSint.bas2.elstat = 1)
151   Then Begin
152     ELangle:= MLSint.ELangle;
153     ELangle_flag:= False;
154   End
155   Else ELangle_flag:= True;
156
157   If (Not MLSint.AZangle_flag) And (MLSint.bas2.azstat = 1) And
158   (Not MLSint.AuxA1_flag)
159   Then Begin
160     Azangle:= MLSint.Azangle + MLSint.AuxA1.AzAlignRun;
161     Azangle_flag:= False;
162   End
163   Else Azangle_flag:= True;
164
165   If (Not MLSint.BAZangle_flag) And (MLSint.bas2.bazstat = 1) And

```

```

166     (MLSint.bas5.bazstat = 1) And (Not MLSint.AuxA4_flag)
167   Then Begin
168     Bazangle:= MLSint.Bazangle - MLSint.AuxA4.BazAlignRun;
169     BAZangle_flag:= False;
170   End
171   Else BAZangle_flag:= True;
172
173   If (Not MLSint.DME_flag) And (MLSint.bas2.dmeestat > 0)
174   Then Begin
175     DMErange:= MLSint.DMErange;
176     DME_flag:= False;
177   End
178   Else DME_flag:= True;
179
180   flag:= ( ELangle_flag Or Elpos_flag) And
181   ( AZangle_flag Or Azpos_flag) And
182   ( Bazangle_flag Or Bazpos_flag) And
183   ( DME_flag Or DMEpos_flag); { set the MLS flag}
184
185   { NOTE: this is magnetic heading}
186   If ( Not MLSint.Bas4_flag) And (Not MLSint.AuxA1_flag)
187   Then Begin
188     Runwayhdg:= MLSint.Bas4.AzMagOr -
189     MLSint.AuxA1.AzAlignRun + 180;
190     If Runwayhdg >= 360
191     Then Runwayhdg:= Runwayhdg - 360;
192     Runwayhdg_flag:= False;
193   End
194   Else Runwayhdg_flag:= True;
195 End;
196 End;
197
198 Procedure ExecMLScommand( command: commandtype);
199
200 Begin
201   ExecMLSreccommand( command);
202 End;
203
204
205 Procedure CloseMLS( MLSdata: MLSdatatype);
206
207 Begin
208   CloseMLSRec;
209 End;
210
211
212
213 Begin
214   With MLSint Do
215     Begin
216       Bas1_flag:= True;
217       Bas2_flag:= True;
218       Bas3_flag:= True;
219       Bas4_flag:= True;
220       Bas5_flag:= True;

```

```
221      Bas6_flag:= True;
222      AuxA1_flag:= True;
223      AuxA2_flag:= True;
224      AuxA3_flag:= True;
225      AuxA4_flag:= True;
226      AuxB_flag:= True;
227      AuxC_flag:= True;
228
229      EAngle_flag:= True;
230      AZangle_flag:= True;
231      BAZangle_flag:= True;
232      DME_flag:= True;
233      discretess_flag:= True;
234      leftclr:= True;
235      rightclr:= True;
236      flag:= True;
237
238      ErrorTime( Bas1_time);
239      ErrorTime( Bas2_time);
240      ErrorTime( Bas3_time);
241      ErrorTime( Bas4_time);
242      ErrorTime( Bas5_time);
243      ErrorTime( Bas6_time);
244      ErrorTime( AuxA1_time);
245      ErrorTime( AuxA2_time);
246      ErrorTime( AuxA3_time);
247      ErrorTime( AuxA4_time);
248      ErrorTime( AuxB_time);
249      ErrorTime( AuxC_time);
250      End;
251 End.
```

```

1 Unit Att;
2
3 {$N+,E+}
4
5 Interface
6
7 Uses MIASglob;
8
9 Procedure InitAtt( Var Attdata: Attdatatype);
10 {*****}
11 {Initialise the Attitude sensor. If the initialising was not succesfull, the
12  error-flag will be true.      Input :-
13                               Output:error}
14 {*****}
15
16
17 Procedure GetAttdata( Var Attdata: Attdatatype);
18 {*****}
19 {Retrieve attitude data from the sensor.
20                               Input :-
21                               Output:Attdata}
22 {*****}
23
24
25 Procedure ExecAttcommand( command: commandtype);
26 {*****}
27 { This procedure receives a command and passes it on to the att device
28  driver.
29                               Input : command
30                               Output: -}
31 {*****}
32
33
34 Procedure CloseAtt( Attdata: Attdatatype);
35 {*****}
36 {Closedown the attitude sensor. Input :-
37                               Output:-}
38 {*****}
39
40
41 Implementation
42
43 Uses AttBeaver;
44
45 Procedure InitAtt( Var Attdata: Attdatatype);
46
47 Var
48     error      :      Boolean;
49
50 Begin
51     error:= True;
52     Attdata.flag:= error;
53
54     If ( attdata.present = 0)
55     Then Exit;
56
57     InitAttTX( error);
58     Attdata.flag:= error;
59 End;
60
61
62 Procedure GetAttdata( Var Attdata: Attdatatype);
63
64 Begin
65     CollectAtt( attdata);
66 End;
67
68
69 Procedure ExecAttcommand( command: commandtype);
70 Begin
71     ExecAtttxcommand( command);
72 End;
73
74
75 Procedure CloseAtt( Attdata: Attdatatype);
76
77 Begin
78     CloseAttTx;
79 End;
80
81 Begin
82 End.

```

```

1 Unit HDG;
2
3 {$N+,E+}
4
5 Interface
6
7 Uses MIAsglob;
8
9 Procedure InitHDG( Var HDGdata: HDGdatatype);
10 {*****}
11 {Initialise the HDG sensor. If the initialising was not succesfull, the
12  error-flag will be true.      Input :-
13                                Output:error}
14 {*****}
15
16
17 Procedure GetHDGdata( Var HDGdata: HDGdatatype);
18 {*****}
19 {Retrieve heading data from the sensor.
20                                Input :-
21                                Output:HDGdata}
22 {*****}
23
24
25 Procedure ExecHDGcommand( command: commandtype);
26 {*****}
27 { This procedure receives a command for the HDG sensor. It passes
28  this command to the HDG device driver.
29                                Input : command
30                                Output: -}
31 {*****}
32
33
34 Procedure CloseHDG( HDGdata: HDGdatatype);
35 {*****}
36 {Closedown the heading sensor.  Input :-
37                                Output:-}
38 {*****}
39
40
41
42 Implementation
43
44 Uses HDGbeaver;
45
46
47 Procedure InitHDG( Var HDGdata: HDGdatatype);
48
49 Var
50     error      :      Boolean;
51
52 Begin
53     error:= True;
54     HDGdata.flag:= error;
55
56     If ( hdgdata.present = 0)
57     Then Exit;
58
59     InitHDGtx( error);
60     HDGdata.flag:= error;
61 End;
62
63
64 Procedure GetHDGdata( Var HDGdata: HDGdatatype);
65
66 Begin
67     CollectHdg( hdgdata);
68 End;
69
70
71 Procedure ExecHDGcommand( command: commandtype);
72
73 Begin
74     ExecHDGtxcommand( command);
75 End;
76
77
78 Procedure CloseHDG( HDGdata: HDGdatatype);
79
80 Begin
81     CloseHDGTX;
82 End;
83
84
85 Begin
86 End.

```



```

1 Unit DGPS;
2
3 {$N+,E+}
4
5 Interface
6
7 Uses MIASglob, crt;
8
9 Procedure InitDGPS( Var DGPSdata: DGPSdatatype);
10 {*****}
11 {Initialise the DGPS sensor. If the initialising was not succesfull, the
12 error-flag will be true.      Input :-
13                               Output:error}
14 {*****}
15
16
17 Procedure GetDGPSdata( Var MLSdata: MLSdatatype; Var DGPSdata: DGPSdatatype);
18 {*****}
19 {Retrieve differential GPS data from the sensor.
20      Input :MLSdata ( ADW's )
21      Output:DGPSdata)
22 {*****}
23
24
25 Procedure CalcDGPS( Var alldata: alldatatype);
26 {*****}
27 { Calculate the corrected ranges for GPS pseudoranges
28      So the record containing differential corrections, are used to
29      correct. The GPS ranges are so corrected for SA etc.
30      Input : alldata, especially DGPS
31      Output: alldata, especially GPS}
32 {*****}
33
34
35 Procedure ExecDGPScommand( command: commandtype);
36 {*****}
37 { This procedure receives a command for the DGPS part of MIAS. The command
38 will be passed on to the DGPS device driver.
39      Input : command
40      Output: -}
41 {*****}
42
43
44 Procedure CloseDGPS( DGPSdata: DGPSdatatype);
45 {*****}
46 {Closedown the differential GPS sensor.
47      Input :-
48      Output:-}
49 {*****}
50
51
52
53 Implementation
54
55 Uses Miscell;

```

```

56
57 Procedure InitDGPS( Var DGPSdata: DGPSdatatype);
58
59 Var
60     x      :      Byte;
61     error:      Boolean;
62
63 Begin
64     error:= True;
65     With Dgpsdata Do
66     Begin
67         For x:= 1 To 32 DO
68             Begin
69                 nabla[x].flag:= True;
70             End;
71     End;
72     DGPSdata.flag:= error;
73 End;
74
75
76 Procedure GetDGPSdata( Var MLSdata: MLSdatatype; Var DGPSdata: DGPSdatatype);
77
78 Const
79     resolution0 =      0.005;
80     resolution1 =      0.01;
81
82 Var
83     currenttime :      Timetype;
84     adress,
85     sv_id :      Integer;
86
87
88 Function ADW_read( ADW: ADWtype; position, bits: Integer): Longint;
89 {*****}
90 { This function takes bits 13 to 18 of the ADW and translates these
91 bits to an adress. This is a normal binary code. See Annex 10 p 150
92      Input : ADW
93      Output: adress}
94 {*****}
95 Var
96     value,
97     mult,
98     x      :      Longint;
99
100 Begin
101     value:= 0;
102     mult:= 1;
103     For x:= 2 To Bits Do
104         mult:= mult * 2;
105
106     For x:= 0 To bits-1 Do
107         Begin
108             value:= value + ADW[ position + x] * mult;
109             mult:= mult Div 2;
110         End;

```

```

111     ADW_read:= value;
112 End;
113
114 Function Twos_complement( ADW: ADWtype; position, bits: Integer): Longint;
115 {*****}
116 { This function takes bits 13 to 18 of the ADW and translates these
117   bits to an adress. This is a normal binary code. See Annex 10 p 150
118     Input : ADW
119     Output: adress}
120 {*****}
121 Var
122     bin_max,
123     value,
124     mult,
125     x      :      Longint;
126
127 Begin
128     value:= 0;
129     bin_max := Round ( Exp ( ( bits - 1 ) * Ln ( 2 ) ) );
130     mult:= bin_max;
131
132     For x:= 0 To bits-1 Do
133     Begin
134         value:= value + ADW[ position + x] * mult;
135         mult:= mult Div 2;
136     End;
137     IF value > bin_max
138     THEN BEGIN
139         value := value - bin_max * 2;
140     END;
141
142     Twos_complement:= value;
143 End;
144
145 Begin
146 End;
147
148
149 Procedure CalcDGPS( Var alldata: alldatatype);
150
151 Var
152     x      :      Integer;
153     h,
154     m,
155     s,
156     s100 :      Word;
157     use_time,
158     update_time : Real;
159     delta_PR    : Double;
160     delayT      : Double;
161     currenttime : timetype;
162
163 Begin
164     Alldata.dgps.flag:= True;
165     delayT      := 5 ; { halve window lengte }

```

```

166
167 With alldata Do
168     For x := 1 to 32 Do
169         With DGPS.nabla[x] Do
170             Begin
171                 If NOT flag Then
172                     Begin
173                         update_time := alldata.GPS.prn[x].rxtime -
174                                     rxtime_dgps
175                                     + delayT;      { time between receiving at
176                                                     ground station and mobile
177                                                     receiver }
178
179                         If Abs( update_time) > 90
180                         Then dgps.nabla[x].flag:= true
181                         Else Begin
182                             delta_PR := order0 + update_time * order1;
183
184                                     { Time update for the
185                                     correction data }
186
187                             With GPS.prn[x] Do
188                                 PR := PR - delta_PR;      { correction of th
189
190                             End;
191                         End;
192                     dgps.flag:= dgps.flag And dgps.nabla[x].flag;
193                 End;
194             End;
195
196 Procedure ExecDGPScommand( command: commandtype);
197
198 Begin
199     ExecDGPSreccommand( command);
200 End;
201
202
203 Procedure CloseDGPS( DGPSdata: DGPSdatatype);
204
205 Begin
206 End;
207
208 Begin
209 End.

```

```

1 Unit User;
2
3 {*****}
4 { This unit provides the necessary routines for inputting and outputting
5   information from and to a user. This user may be a person, or an FMS.}
6 {*****}
7
8 Interface
9
10 Uses MiasGlob;
11
12 Procedure InitUser( MIASlogname: string);
13 {*****}
14 { Only here for compatibility}
15 {*****}
16
17
18 Procedure SendUserMessage( Message: commandtype);
19 {*****}
20 { display a message commandtype to the user}
21 {
22   Input : message
23   Output: message on the outputdevice}
24 {*****}
25
26 Procedure GetUserMessage( Var Message: commandtype);
27 {*****}
28 { display a message commandtype to the user}
29 {
30   Input : message from the inputdevice
31   Output: message }
32 {*****}
33
34 Procedure SenduserFlags( Message: commandtype);
35 {*****}
36 { This procedure sends flags to the outputdevice}
37 {
38   Input : message
39   Output: message on the outputdevice}
40 {*****}
41 Procedure SaveEquipmentMessage( message: commandtype);
42 {*****}
43 { save the messages from equipment in a file}
44 {
45   Input : message
46   Output: message to a file}
47 {*****}
48
49 Procedure CloseUser;
50 {*****}
51 { Only for compatibility. Closes the outputfile 'MIAS.OUT'}
52 {
53   Input : -
54   Output: -}
53 {*****}
54 {*****}
55

```

```

56
57 Implementation
58
59 Uses Key_Const;
60
61 Procedure InitUser( MIASlogname: string);
62
63 Begin
64   OpenIn_OutputDev( MIASlogname);
65 End;
66
67 Procedure SendUserMessage( Message: commandtype);
68
69 Begin
70   SendMessage( message);
71 End;
72
73
74 Procedure GetUserMessage( Var Message: commandtype);
75
76 Begin
77   GetMessage( message);
78 End;
79
80 Procedure SenduserFlags( Message: commandtype);
81
82 Begin
83   SendFlags( message);
84 End;
85
86 Procedure SaveEquipmentMessage( message: commandtype);
87
88 Begin
89   SaveMessage( message);
90 End;
91
92 Procedure CloseUser;
93
94 Begin
95   CloseIn_outputDev;
96 End;
97
98 End.

```

```

1 Unit Poscalc;
2
3 {$N+,E+}
4
5 Interface
6
7 Uses MIASGLOB;
8
9 Var
10   MLSantposition      :      positiontype;
11
12 Procedure CalcHybridPos( Var alldata: alldatatype; allowed_error: Double;
13   Var position: positiontype);
14 {*****}
15 { This procedure calculates the hybrid position of an aircraft using
16   GPS and MLS information.
17   Input : alldata
18   Output: position}
19 {*****}
20
21
22 procedure Convert_Pos_to_ECEF( Var position: positiontype );
23 {*****}
24 (* Procedure for conversion of the receiver's position from
25 (* <lat,long,alt> coordinates to ECEF-coordinates X,Y and Z.
26 (* (dimension <rad,rad,m> -----> dimension <m,m,m>
27 (*
28 (* This procedure needs:
29 (* - position (WGS-84 coordinates)
30 (* (latitude, logitude, altitude) [ rad,rad, m]
31 (*
32 (* This procedure supplies:
33 (* - position (ECEF - coordinates)
34 (* ( X, Y, Z) [ m, m, m]
35 (*
36 (* G.L. van Eendenburg, December 1988
37 {*****}
38
39
40 procedure Convert_Pos_to_WGS( Var position: positiontype);
41 {*****}
42 (* Procedure for conversion of the receiver's ECEF-coordinates
43 (* (X,Y,Z) to <lat,long,alt> - coordinates.
44 (* dimension <m,m,m> -----> dimension <rad,rad,m>
45 (* This is an iterative procedure
46 (*
47 (* This procedure needs:
48 (* - position (ECEF - coordinates)
49 (* ( X, Y, Z) [ m, m, m]
50 (*
51 (* This procedure supplies:
52 (* - position (WGS-84 coordinates)
53 (* (latitude, logitude, altitude) [ rad,rad, m]
54 (* - procedure status
55 (*
56 (* G.L. van Eendenburg, December 1988 *)
57 {adjusted to specific MIAS needs R.C. Meijer, August 1992}
58 {*****}
59
60
61 Implementation
62
63 USES MATRIX, MATHX, crt, user;
64
65 Const
66   flattening = 3.3528106E-3; {flattening of the earth }
67   earthAxis = 6.378137E+6; {long earth axis }
68   earthEccentricitySqr = flattening * (2 - flattening); { e^2 !! }
69   pi = 3.1415926535897932385;
70   tweepi = 2 * pi;
71   c = 2.99792458E8; {[m/s] speed of light}
72
73 Procedure Convert_Pos_to_ECEF( Var position: positiontype );
74
75 Var
76   EarthRadiusN : Double;
77
78 Begin
79   EarthRadiusN := EarthAxis / sqrt( 1 - EarthEccentricitySqr *
80     sqrt(sin(position.WGS84lat)) );
81   position.X := (EarthRadiusN + position.WGS84alt) *
82     cos(position.WGS84lat) * cos(position.WGS84lon);
83   position.Y := (EarthRadiusN + position.WGS84alt) *
84     cos(position.WGS84lat) * sin(position.WGS84lon);
85   position.Z := (EarthRadiusN * ( 1 - EarthEccentricitySqr ) +
86     position.WGS84alt ) * sin(position.WGS84lat);
87 End; (* of the procedure Convert_Pos_To_ECEF *)
88
89
90 Procedure Convert_Pos_to_WGS( Var position: positiontype);
91
92 Const
93   smallValue = 1E-10; { small value to check if user is at a pole, }
94   { requested accuracy of NewtonRaphson etc. }
95   verySmall = 1E-15; { small value to prevent denominator = 0 }
96   maxIterAllowed = 25; { maximum number of iterations allowed }
97
98 Var
99   numberOfRuns : byte;
100   C,func1,Delta_lat : Double;
101
102 Begin
103   With position Do
104     Begin
105       { secures against crash on Poles }
106       If (Abs( X ) < smallValue) And
107         (Abs( Y ) < smallValue)
108       Then Begin { user is at or close to North/South Pole }
109         If ( Z > 0 ) Then
110           WGS84lat := pi/2 { North Pole }

```

```

111     Else
112         WGS84lat := -pi/2;           { South Pole }
113
114
115         { EXPERIMENTAL }
116         { the value of the longitude is calculated }
117         { as zero, but with help of the course }
118         { this value must be defined yet !!!!!!! }
119         WGS84lon := 0;
120         WGS84alt := Abs( Z ) - (EarthAxis *
121             sqrt(1 - earthEccentricitySqr) );
122
123     End;
124
125         { calculating the longitude }
126         { the ellipsoid is a perfect circle }
127         { in the equator plane. }
128     If ( Abs( X ) < verySmall ) then
129     Begin
130         { protects arctan function against zero X }
131         If X >= 0 then X := verySmall;
132         if X < 0 then X := -verySmall;
133     End;
134     WGS84lon := arctan( Y / X );
135         { correction of the arctan function: changes }
136         { range from -pi/2 -> pi/2 to -pi -> pi. }
137     If ( ( X < 0 ) And ( Y > 0 ) )
138     Then WGS84lon := WGS84lon + pi;
139
140     If ( ( X < 0 ) And ( Y < 0 ) )
141     Then WGS84lon := WGS84lon - pi;
142
143         { calculation of the latitude with Newton-Raphson }
144     C := Sqrt( Sqr( X ) + Sqr( Y ) );
145     WGS84lat := arctan( Z / C );
146     numberOfRuns := 1;
147     Delta_lat := 10;           { initial value }
148
149     While ( Abs( Delta_lat ) > smallValue ) And
150         ( numberOfRuns < maxIterAllowed ) Do
151     Begin
152         If WGS84lat = 0.5*pi
153         Then WGS84lat := 0.5*pi - smallValue;
154
155         func1 := 1 / Sqrt( 1 - earthEccentricitySqr *
156             Sqr( sin( WGS84lat ) ) );
157
158         Delta_lat := ( Z - C * sin( WGS84lat ) /
159             cos( WGS84lat ) + earthEccentricitySqr *
160             EarthAxis * func1 * sin( WGS84lat ) ) /
161             ( C / Sqr( cos( WGS84lat ) ) - EarthAxis *
162             cos( WGS84lat ) * earthEccentricitySqr *
163             func1 * ( 1 + earthEccentricitySqr *
164             Sqr( func1 * sin( WGS84lat ) ) ) );
165
166         WGS84lat := WGS84lat + Delta_lat;

```

```

166         numberOfRuns := numberOfRuns + 1;
167     End;
168
169     If numberOfRuns >= maxIterAllowed
170     Then flag := True;           { failure in Newton Raphson }
171
172         { calculation of the altitude }
173
174         { see "GPS" }
175         { x = [ Na + ha ] cos(lat)cos(lon) }
176         { x / [ cos(lat)cos(lon) ] = Na + ha }
177         { x / [ cos(lat)cos(lon) ] - Na = ha }
178         { with Na = ae / sqrt( 1 - ) }
179         { sqrt(e)sqr( sin(lat)) ] }
180
181     WGS84alt := X /
182         ( cos( WGS84lat ) * cos( WGS84lon ) ) -
183         earthAxis /
184         Sqrt( 1 - earthEccentricitySqr * Sqr( sin( WGS84lat ) ) );
185
186     End; { end with position }
187 End; { of procedure Convert_Pos_To_WGS }
188
189
190
191 CONST
192
193     ae = 6.378137E+0006; {metres} {The equatorial radius}
194     be = 6.356752E+0006; {metres} {The polar distance }
195
196     {-----}
197     {MaxNoOfGPSequations is the maximum number of linearised GPS-equations }
198     {which is the same as the maximum number of sattelites used. }
199
200     MaxNoOfGPSequations = 7;
201
202     {-----}
203     {MaxNoOfMLSequations is the maximum number of linearised MLS-equations.}
204     {The three equations are; the azimuthequation, the elevationequation }
205     {and the DME-equation. }
206
207     MaxNoOfMLSequations = 3;
208     maxiter= 25;
209
210 TYPE CHARROW = PACKED ARRAY[1..10] OF CHAR;
211
212 GPSROW = ARRAY[1..MaxNoOfGPSequations] OF INTEGER;
213 MLSROW = ARRAY[1..MaxNoOfMLSequations] OF INTEGER;
214
215 RANGE = ARRAY[1..MaxNoOfGPSequations] OF DOUBLE;
216
217 MLS = ARRAY[1..MaxNoOfMLSequations] OF DOUBLE;
218
219 MLSBLOCK = ARRAY[1..MaxNoOfMLSequations,1..3] OF DOUBLE;
220

```

```

221 WGS84 = RECORD
222     longitude,latitude,height:DOUBLE;
223     hemilong,hemilat :CHARROW;
224     END;
225
226 {-----}
227 {In ECEF and LOCAL; ARRAY[1] = x-coordinate, }
228 {     ARRAY[2] = y-coordinate, }
229 {     ARRAY[3] = z-coordinate. }
230 {     ARRAY[4] = clockerror (dR = c*dT) }
231
232 ECEF = ARRAY[1..4] OF DOUBLE;
233
234 LOCAL = ARRAY[1..4] OF DOUBLE;
235
236 {-----}
237
238 SATECEF = ARRAY[1..MaxNoOfGPSequations] OF ECEF;
239 SATLOCAL = ARRAY[1..MaxNoOfGPSequations] OF LOCAL;
240
241 VAR {-----}
242 {NumberOfGPSequations describes the total amount of GPS-equations and }
243 {NumberOfMLSequations describes the total amount of MLS-equations used in}
244 {this program. TotalOfEquations is NumberOfGPSequations + }
245 {NumberOfMLSequations. }
246
247 NumberOfGPSequations,
248 NumberOfMLSequations,
249 TotalOfEquations :INTEGER;
250
251 {-----}
252 {First eccentricity of the earth }
253
254 e2,
255
256 {-----}
257 {The Nort_angle is the angle between the true north of the earth and the }
258 {y-axis of the MLS-Coordinate system (Local Reference). }
259
260 North_angle,
261
262 {-----}
263 {The rollangle, pitchangle, and the headingangle of the airplane. }
264
265 Rollangle,
266 Pitchangle,
267 Headingangle :DOUBLE;
268
269 {-----}
270 {The ECEF-Coordinates of the -The estimated ECEF-position of the airborne}
271 {     GPS-antenna, }
272 {     -MLS-datumpoint (Datumpoint_ecef), }
273 {     -The real (RealGPS_ecef) ECEF-position of }
274 {     the airborne GPS-antenna, }
275 {     -The real (RealMIAS_ecef) ECEF-position of }
276 {
277 {
278 {
279 {
280 {
281
282 Est_ecef,
283 Datumpoint_ecef,
284 Fix_ecef,
285 Variance_ecef :ECEF;
286
287 {-----}
288 {The WGS84-Coordinates of the -MLS-datumpoint (Datumpoint_wgs84), }
289 {     -The real (RealGPS_wgs84) WGS84-position of }
290 {     the airborne GPS-antenna, }
291 {     -The real (RealMIAS_wgs84) WGS84-position }
292 {     of the airborne MLS-antenna. }
293
294 EstWgs84,
295 Datumpoint_wgs84 :wgs84;
296
297 {-----}
298 {The MLS-Coordinates of - the estimated position of the airborne MLS- }
299 {     antenna, }
300 {     - the azimuthantennaposition (ground), }
301 {     - the elevationantennaposition (ground), }
302 {     - the DME-antennaposition (ground), }
303 {     - the real (Real_local) position of the }
304 {     airborne MLS-antenna, }
305 {     - the variance in the estimated position of the }
306 {     airborne MLS-antenna in local coordinates, }
307 {     - the final fix of a certain defined point within }
308 {     the aircraft. }
309
310 Est_local,
311 Azi_local,
312 Ele_local,
313 DME_local,
314 Variance_local,
315 Fix_local,
316
317 {-----}
318 {The local coordinates of the Antenna_Zerovector (which is the vector }
319 {between the airborne MLS-antenna as origin and the airborne GPS-antenna }
320 {when roll, pitch and heading are zero) and the GPSPosition_Zerovector }
321 {(which is the vector between the airborne GPS-antenna as origin and a }
322 {certain defined point within the airplane (for example the wheels)) and }
323 {the MIASPosition_Zerovector (which is the vector between the airborne }
324 {MLS-antenna as origin and a certain defined point within the airplane). }
325
326 Antenna_Zerovector,
327 GPSPosition_Zerovector,
328 MIASPosition_Zerovector :LOCAL;
329
330 {-----}

```

```

331 {GPS_eq describes which GPS-equations are used in the algorithm and } 386
332 {MLS_eq describes which MLS-equations are used in the algorithm in order } 387
333 {to calculate the position with a subset of the available equations } 388
334 {needed for integrity. } 389
335 390
336 GPS_eq :GPSROW; 391
337 MLS_eq :MLSROW; 392
338 393
339 {-----} 394
340 {The measured pseudoranges (R_meas), the estimated ranges (R_est) and the} 395
341 {ranges between the airborne MLS-antenna and the satellites (R_adjust). } 396
342 397
343 R_meas, 398
344 R_est, 399
345 R_adjust, 400
346 401
347 {-----} 402
348 {The variances of the GPS-pseudoranges and the DGPS-pseudoranges. } 403
349 404
350 Var_Rgps, 405
351 Var_Rdgps :RANGE; 406
352 407
353 {-----} 408
354 {The measured MLS-parametres (azimuthangle, elevationangle, DME-distance)} 409
355 410
356 MLS_meas, 411
357 412
358 {-----} 413
359 {The variances of the MLS-parametres. } 414
360 415
361 Var_MLS :MLS; 416
362 417
363 {-----} 418
364 {The calculated MLS-data needed for the MIASmatrix which are; } 419
365 { -The estimated azimuthangle, (MLS_data[x,1]) } 420
366 { -The estimated elevationangle, (MLS_data[x,1]) } 421
367 { -The estimated DME_distance, (MLS_data[x,1]) } 422
368 { -The angle azimuthangle accent, (MLS_data[x,2]) } 423
369 { -The angle elevationangle accent, (MLS_data[x,2]) } 424
370 { -The angle DMEangle 1, (MLS_data[x,2]) } 425
371 { -The azimuthdistance, (MLS_data[x,3]) } 426
372 { -The elevationdistance, (MLS_data[x,3]) } 427
373 { -The angle DMEangle 2, (MLS_data[x,3]) } 428
374 429
375 MLS_data :MLSBLOCK; 430
376 mls1, 431
377 mls2, 432
378 mls3, 433
379 gps1, 434
380 gps2, 435
381 gps3, 436
382 gps4, 437
383 gps5, 438
384 gps6, 439
385 gps7 :Byte; 440

```

```

{-----}
{The satellitepositions in ECEF (Satecef) and in Local Reference }
{((satlocal)). }

Sat_ecef :SATECEF;
Sat_local :SATLOCAL;

{-----}
{The matrix with the variances of the GPS-pseudoranges on the diagonal }
{GPSVARIANCEmatrix) and the matrix with the variances of the }
{DGPS-pseudoranges and the variances of the azimuthangle, elevationangle }
{and the DME-distance on the diagonal (MIASVARIANCE matrix). }

GPSVARIANCEmatrix,
MIASVARIANCEmatrix,

{-----}
{The GPS-matrix and the MIAS-matrix. }

GPSmatrix,
MIASmatrix,

{-----}
{The GPS-vector and the MIAS-vector. }

GPSvector,
MIASvector,

{-----}
{The weighed Least-Square matrix. }

WLSqMatrix,
miasweighmatrix,

{-----}
{The vector with the x-, y-, z-, and clockerror-corrections for the }
{estimated position. }

DeltaVector :TNMATRIX;

{-----}
{Boolean values indicating if GPS- and MLS-signals are present or not. }

GPSavailable,
MLSavailable :BOOLEAN;

{-----}

Error :BYTE;
present :Record
att,
hdg,
gps,
mls,

```

```

441          dgps      :      Byte;
442      End;
443
444      Convergence_Error      :DOUBLE;
445      iter: integer;
446
447
448  PROCEDURE Show_Matrix(TotalOfRows,
449                        TotalOfCols:INTEGER;
450                        Matrix      :TNMATRIX);
451  VAR i,j:INTEGER;
452
453  {*****}
454  (-Input ; TotalOfRows (= Total of rows in Matrix) )
455  {      TotalOfCols (= Total of columns in Matrix) }
456  {      Matrix }
457  (-Output; - )
458  { }
459  {This procedure shows the values of the coefficients of Matrix on screen. }
460  {*****}
461
462  BEGIN
463    FOR i:=1 TO TotalOfRows DO
464      BEGIN
465        FOR j:=1 TO TotalOfCols DO
466          write(matrix[i,j]:16,' ');
467          writeln;
468        END;
469        writeln;
470      END;{PROCEDURE Show_Matrix}
471
472
473
474  PROCEDURE Clean_Matrix(TotalOfRows,
475                        TotalOfCols:INTEGER;
476                        VAR Matrix      :TNMATRIX);
477  VAR i,j:INTEGER;
478
479  {*****}
480  (-Input ; TotalOfRows (= Total of rows in Matrix) )
481  {      TotalOfCols (= Total of columns in Matrix) }
482  (-Output; Matrix )
483  { }
484  {This procedure sets all coefficients in Matrix to zero. }
485  {*****}
486  BEGIN
487    FOR i:=1 TO TotalOfRows DO
488      BEGIN
489        FOR j:=1 TO TotalOfCols DO Matrix[i,j]:=0;
490      END;
491    END;{PROCEDURE Clean_Matrix}
492
493
494
495  FUNCTION RowColumnMult(LineLengthM1,

```

```

496      RowM1,
497      ColM2      :INTEGER;
498      Matrix1,
499      Matrix2      :TNMATRIX):DOUBLE;
500  VAR i :INTEGER;
501      Sum:DOUBLE;
502
503  {*****}
504  (-Input ; LineLengthM1 (= The length of a line in Matrix1 (Total of columns )
505  {      in Matrix1)) )
506  {      RowM1 (= The rownumber of Matrix1) )
507  {      ColM2 (= The columnnumber of Matrix2) )
508  {      Matrix1 )
509  {      Matrix2 )
510  (-Output; RowColumnMult )
511  { }
512  {This function calculates the product of a row in Matrix1 (RowM1) and a )
513  {column in Matrix2 (ColM2). }
514  {*****}
515
516  BEGIN
517    Sum:=0;
518    FOR i:=1 TO LineLengthM1 DO
519      Sum:=Sum+Matrix1[RowM1,i]*Matrix2[i,ColM2];
520    RowColumnMult:=Sum;
521  END;{FUNCTION RowColumnMult}
522
523
524
525  PROCEDURE Matrix_Mult(TotalOfRowsM1,
526                        TotalOfColsM1,
527                        TotalOfColsM2 :INTEGER;
528                        Matrix1,
529                        Matrix2      :TNMATRIX;
530                        VAR SolMatrix :TNMATRIX);
531  VAR i,j:INTEGER;
532
533  {*****}
534  (-Input ; TotalOfRowsM1 (= Total of rows in Matrix1) )
535  {      TotalOfColsM1 (= Total of columns in Matrix1) )
536  {      TotalOfColsM2 (= Total of columns in Matrix2) )
537  {      Matrix1 )
538  {      Matrix2 )
539  (-Output; SolMatrix )
540  { }
541  {This procedure calculates SolMatrix (SolutionMatrix) = Matrix1*Matrix2. }
542  {*****}
543
544  BEGIN
545    FOR i:=1 TO TotalOfRowsM1 DO
546      BEGIN
547        FOR j:=1 TO TotalOfColsM2 DO
548          SolMatrix[i,j]:=RowColumnMult(TotalOfColsM1,i,j,Matrix1,Matrix2);
549        END;
550      END;{PROCEDURE Matrix_Mult}

```



```

551
552
553
554 PROCEDURE Transpose(TotalOfRows,
555                      TotalOfCols:INTEGER;
556                      Matrix      :TNMATRIX;
557                      VAR SolMatrix :TNMATRIX);
558 VAR i,j:INTEGER;
559
560 {*****}
561 {-Input ; TotalOfRows (= Total of rows in Matrix) }
562 {      TotalOfCols (= Total of columns in Matrix }
563 {      Matrix      }
564 {-Output; SolMatrix }
565 { }
566 {This procedure calculates SolMatrix (SolutionMatrix) = The transpose of }
567 {Matrix. }
568 {*****}
569
570 BEGIN
571   FOR i:=1 TO TotalOfRows DO
572     BEGIN
573       FOR j:=1 TO TotalOfCols DO SolMatrix[j,i]:=Matrix[i,j];
574     END;
575   END;{PROCEDURE Transpose}
576
577
578
579 PROCEDURE WGS84_to_ECEF(POSwgs84:WGS84;
580                      VAR POSecef :ECEF);
581 VAR Na:DOUBLE;
582
583 {*****}
584 {-Input ; POSwgs84 (the position in WGS84) }
585 {-Output; POSecef (the position in ECEF) }
586 { }
587 {This procedure calculates a position given in WGS84-coordinates to a }
588 {position given in ECEF-coordinates. }
589 {*****}
590
591 BEGIN
592   WITH POSwgs84 DO
593     BEGIN
594       {-----}
595       {Conversion from degrees into radials depending upon hemisphere. }
596
597       IF hemilong = 'east' THEN longitude:= longitude*pi/180
598       ELSE {hemilong = 'west'} longitude:=-longitude*pi/180;
599       IF hemilat = 'north' THEN latitude:= latitude*pi/180
600       ELSE {hemilat = 'south'} latitude:=-latitude*pi/180;
601
602       {-----}
603       {Take into account the flexure of the earth. }
604
605       Na:=ae/sqrt(1-e2*sqr(sin(latitude)));

```

```

606
607 {-----}
608 {Calculate the ECEF-coordinates. }
609
610   POSecef[1]:=(Na+height)*cos(latitude)*cos(longitude);
611   POSecef[2]:=(Na+height)*cos(latitude)*sin(longitude);
612   POSecef[3]:=(Na*(1-e2)+height)*sin(latitude);
613   END;
614 END;{PROCEDURE WGS84_to_ECEF}
615
616
617
618 PROCEDURE ECEF_to_WGS84(POSecef :ECEF;
619                      VAR POSwgs84:WGS84);
620
621 VAR Na,difference,oldlatitude:DOUBLE;
622
623 {*****}
624 {-Input ; POSecef (the position in ECEF) }
625 {-Output; POSwgs84 (the position in WGS84) }
626 { }
627 {This procedure calculates a position given in ECEF-coordinates to a position}
628 {given in WGS84-coordinates. }
629 {*****}
630
631 BEGIN
632
633   WITH POSwgs84 DO
634     BEGIN
635       {-----}
636       {Calculate the longitude. }
637
638       IF POSecef[1] = 0 THEN longitude:=pi/2;
639       IF POSecef[2] = 0 THEN
640         BEGIN
641           hemilong:='gwmmeridian';
642           IF POSecef[1] >= 0 THEN longitude:=0
643           ELSE longitude:=pi;
644         END
645       ELSE
646         BEGIN
647           IF POSecef[2] > 0 THEN
648             BEGIN
649               hemilong:='east';
650               IF POSecef[1] > 0 then longitude:= arctan(POSecef[2]/POSecef[1])
651               ELSE longitude:= pi+arctan(POSecef[2]/POSecef[1]);
652             END
653           ELSE
654             BEGIN
655               hemilong:='west';
656               IF POSecef[1] > 0 then longitude:=-arctan(POSecef[2]/POSecef[1])
657               ELSE longitude:= pi-arctan(POSecef[2]/POSecef[1]);
658             END;
659           END;
660

```

```

661 {-----}
662 {Calculate the latitude.}
663
664 IF POSecef[3] = 0 THEN hemilat:='equator '
665 ELSE IF POSecef[3] > 0 THEN hemilat:='north '
666 ELSE hemilat:='south '
667
668 IF (POSecef[1] = 0) AND (POSecef[2] = 0) THEN latitude:=pi/2
669 ELSE
670 BEGIN
671 latitude:=arctan(POSecef[3]/sqrt(sqr(POSecef[1])+sqr(POSecef[2])));
672 difference:=1;
673 WHILE difference > 1E-0010*pi/180 DO {accuracy of 1E-0010 degrees}
674 BEGIN
675 oldlatitude:=latitude;
676 latitude :=arctan(POSecef[3]/(sqrt(sqr(POSecef[1])+sqr(POSecef[2]))-
677 e2*cos(latitude)*
678 ae/sqrt(1-e2*sqr(sin(latitude)))));
679 difference :=abs(oldlatitude-latitude);
680 END;
681 latitude:=abs(latitude);
682 END;
683
684 {-----}
685 {Calculate the height.}
686
687 Na :=ae/sqrt(1-e2*sqr(sin(latitude)));
688 height:=(sqrt(sqr(POSecef[1])+sqr(POSecef[2]))/cos(latitude))-Na;
689
690 {-----}
691 {Conversion from radials into degrees.}
692
693 longitude:=longitude*180/pi;
694 latitude :=latitude*180/pi;
695 END;
696 END;(PROCEDURE ECEF_to_WGS84)
697
698
699
700 PROCEDURE Local_to_ECEFOrientation(ORIENTlocal:LOCAL;
701 POSwgs84 :WGS84;
702 Northangle :DOUBLE;
703 VAR ORIENTecef :ECEF);
704
705 VAR matrix1,matrix2,matrix3,matrix4,matrx1,matrx2,solmatrix,
706 vector,solvector :TNMATRIX;
707
708 {*****}
709 {-Input ; ORIENTlocal (a vector described in local coordinates)}
710 {- POSwgs84 (the position of the origin of the local system in WGS84-}
711 {- coordinates)}
712 {- Northangle (the angle between the true north and the y-axis of the}
713 {- local system)}
714 {-Output; ORIENTecef (a vector described in a system with the same origin as}
715 {- the local system but with the axes orientated as the }
716 { ECEF-axes)}
717 {
718 {This procedure converts a vector described in local coordinates to a vector}
719 {described in a system with the same origin as the local system but with the}
720 {x-, y-, and z-axes orientated as the ECEF-system.}
721 {*****}
722
723 BEGIN
724 WITH POSwgs84 DO
725 BEGIN
726 {-----}
727 {Conversion from degrees into radials of the point of the origin.}
728
729 longitude :=longitude *pi/180;
730 latitude :=latitude *pi/180;
731 Northangle :=Northangle*pi/180;
732
733 {-----}
734 {Calculate the LONGITUDEmatrix.}
735
736 matrix1[1,1]:= cos(longitude);
737 matrix1[1,2]:=-sin(longitude);
738 matrix1[1,3]:= 0;
739 matrix1[2,1]:= sin(longitude);
740 matrix1[2,2]:= cos(longitude);
741 matrix1[2,3]:= 0;
742 matrix1[3,1]:= 0;
743 matrix1[3,2]:= 0;
744 matrix1[3,3]:= 1;
745
746 {-----}
747 {Calculate the LATITUDEmatrix.}
748
749 matrix2[1,1]:= cos(latitude);
750 matrix2[1,2]:= 0;
751 matrix2[1,3]:=-sin(latitude);
752 matrix2[2,1]:= 0;
753 matrix2[2,2]:= 1;
754 matrix2[2,3]:= 0;
755 matrix2[3,1]:= sin(latitude);
756 matrix2[3,2]:= 0;
757 matrix2[3,3]:= cos(latitude);
758
759 {-----}
760 {Calculate the NORTHANGLEmatrix.}
761
762 matrix3[1,1]:= 1;
763 matrix3[1,2]:= 0;
764 matrix3[1,3]:= 0;
765 matrix3[2,1]:= 0;
766 matrix3[2,2]:= cos(Northangle);
767 matrix3[2,3]:=-sin(Northangle);
768 matrix3[3,1]:= 0;
769 matrix3[3,2]:= sin(Northangle);
770 matrix3[3,3]:= cos(Northangle);

```

```

771
772 {-----}
773 {Rename the axes (RENAMEmatrix).}
774
775 matrix4[1,1]:=0; matrix4[1,2]:=0; matrix4[1,3]:=1;
776 matrix4[2,1]:=1; matrix4[2,2]:=0; matrix4[2,3]:=0;
777 matrix4[3,1]:=0; matrix4[3,2]:=1; matrix4[3,3]:=0;
778
779 {-----}
780 {Write the inputparametre ORIENTlocal to vector in order to use the }
781 {procedure Matrix_Mult.}
782
783 vector[1,1]:=ORIENTlocal[1];
784 vector[2,1]:=ORIENTlocal[2];
785 vector[3,1]:=ORIENTlocal[3];
786
787 {-----}
788 {LONGITUDEmatrix*LATITUDEmatrix*NORTHANGLEmatrix*RENAMEmatrix = solmatrix}
789
790 Matrix_Mult(3,3,3,matrix3,matrix4,matrx1);
791 Matrix_Mult(3,3,3,matrix2,matrx1,matrx2);
792 Matrix_Mult(3,3,3,matrix1,matrx2,solmatrix);
793
794 {-----}
795 {solmatrix*vector = solvector.}
796
797 Matrix_Mult(3,3,1,solmatrix,vector,solvector);
798
799 {-----}
800 {Convert the solvector to ORIENTcefe.}
801
802 ORIENTcefe[1]:=solvector[1,1];
803 ORIENTcefe[2]:=solvector[2,1];
804 ORIENTcefe[3]:=solvector[3,1];
805 END;
806 END;{PROCEDURE Local_to_ECEFOrientation}
807
808
809
810 PROCEDURE ECEF_to_LocalOrientation(ORIENTcefe :ECEF;
811                                     POSwgs84 :WGS84;
812                                     Northangle :DOUBLE;
813                                     VAR ORIENTlocal:LOCAL);
814
815 VAR matrix1,matrix2,matrix3,matrix4,matrx1,matrx2,solmatrix,
816     vector,solvector :TNMATRIX;
817
818 {*****}
819 {-Input ; ORIENTcefe (a vector described in a system with the same origin as }
820 { the local system but with the axes orientated as in }
821 { ECEF }
822 { POSwgs84 (the position of the origin of the local system in WGS84) }
823 { Northangle (the angle between the true north and the y-axis of the }
824 { local system) }
825 {-Output; ORIENTlocal (a vector described in local coordinates) }

```

```

826 {
827 {This procedure converts a vector described by a system with a origin the same}
828 {as the local system but with axes orientated like the ECEF-system to a }
829 {vector described by that local system.}
830 {*****}
831
832 BEGIN
833     WITH POSwgs84 DO
834     BEGIN
835         {-----}
836         {conversion from degrees into radials of the point of the origin.}
837
838         longitude :=longitude *pi/180;
839         latitude :=latitude *pi/180;
840         Northangle :=Northangle*pi/180;
841
842         {-----}
843         {Rename the axes (RENAMEmatrix).}
844
845         matrix1[1,1]:= 0; matrix1[1,2]:= 1; matrix1[1,3]:= 0;
846         matrix1[2,1]:= 0; matrix1[2,2]:= 0; matrix1[2,3]:= 1;
847         matrix1[3,1]:= 1; matrix1[3,2]:= 0; matrix1[3,3]:= 0;
848
849         {-----}
850         {Calculate the NORTHANGLEmatrix.}
851
852         matrix2[1,1]:= 1;
853         matrix2[1,2]:= 0;
854         matrix2[1,3]:= 0;
855         matrix2[2,1]:= 0;
856         matrix2[2,2]:= cos(Northangle);
857         matrix2[2,3]:= sin(Northangle);
858         matrix2[3,1]:= 0;
859         matrix2[3,2]:= -sin(Northangle);
860         matrix2[3,3]:= cos(Northangle);
861
862         {-----}
863         {Calculate the LATITUDEmatrix.}
864
865         matrix3[1,1]:= cos(latitude);
866         matrix3[1,2]:= 0;
867         matrix3[1,3]:= sin(latitude);
868         matrix3[2,1]:= 0;
869         matrix3[2,2]:= 1;
870         matrix3[2,3]:= 0;
871         matrix3[3,1]:= -sin(latitude);
872         matrix3[3,2]:= 0;
873         matrix3[3,3]:= cos(latitude);
874
875         {-----}
876         {Calculate the LONGITUDEmatrix.}
877
878         matrix4[1,1]:= cos(longitude);
879         matrix4[1,2]:= sin(longitude);
880         matrix4[1,3]:= 0;

```

```

881 matrix4[2,1]:=-sin(longitude);
882 matrix4[2,2]:= cos(longitude);
883 matrix4[2,3]:= 0;
884 matrix4[3,1]:= 0;
885 matrix4[3,2]:= 0;
886 matrix4[3,3]:= 1;
887
888 {-----}
889 {Write the inputparametre ORIENTecef to vector in order to use the }
890 {procedure Matrix_Mult. }
891
892 vector[1,1]:=ORIENTecef[1];
893 vector[2,1]:=ORIENTecef[2];
894 vector[3,1]:=ORIENTecef[3];
895
896 {-----}
897 {RENAMEmatrix*NORTHANGLEmatrix*LATITUDEmatrix*LONGITUDEmatrix = solmatrix}
898
899 Matrix_Mult(3,3,3,matrix3,matrix4,matrx1);
900 Matrix_Mult(3,3,3,matrix2,matrx1,matrx2);
901 Matrix_Mult(3,3,3,matrix1,matrx2,solmatrix);
902
903 {-----}
904 {solmatrix*vector = solvector. }
905
906 Matrix_Mult(3,3,1,solmatrix,vector,solvector);
907
908 {-----}
909 {Convert the solvector to ORIENTlocal. }
910
911 ORIENTlocal[1]:=solvector[1,1];
912 ORIENTlocal[2]:=solvector[2,1];
913 ORIENTlocal[3]:=solvector[3,1];
914 END;
915 END;{PROCEDURE ECEF_to_LocalOrientation}
916
917
918
919 PROCEDURE LocalAircraft_to_LocalMLSOrientation(AircraftLocal:LOCAL;
920 Northangle :DOUBLE;
921 VAR MLSLocal :LOCAL);
922 VAR matrix,vector,solvector:TNMATRIX;
923
924 {*****}
925 {-Input ; AircraftLocal (The coordinates of a vector defined within the }
926 { aircraft) }
927 { Northangle (the angle between the true north and the y-axis of the }
928 { MLS Local Reference system) }
929 { MLSLocal (The coordinates of a vector defined within the aircraft }
930 { and orientated as the MLS Local Reference system. }
931 { }
932 {This procedure converts a vector defined in a coordinatesystem in the }
933 {aircraft to a coordinatesystem orientated as the MLS Local Reference system.}
934 {*****}
935

```

```

936 BEGIN
937 {-----}
938 {Calculate the NORTHANGLEmatrix. }
939
940 matrix[1,1]:= 1;
941 matrix[1,2]:= 0;
942 matrix[1,3]:= 0;
943 matrix[2,1]:= 0;
944 matrix[2,2]:= cos(Northangle);
945 matrix[2,3]:= sin(Northangle);
946 matrix[3,1]:= 0;
947 matrix[3,2]:=-sin(Northangle);
948 matrix[3,3]:= cos(Northangle);
949
950 {-----}
951 {Write the inputparameter AircraftLocal to vector in order to use the }
952 {procedure Matrix_Mult. }
953
954 vector[1,1]:=AircraftLocal[1];
955 vector[2,1]:=AircraftLocal[2];
956 vector[3,1]:=AircraftLocal[3];
957
958 {-----}
959 {NORTHANGLEmatrix*vector = solvector. }
960
961 Matrix_Mult(3,3,1,matrix,vector,solvector);
962
963 {-----}
964 {Convert the solvector to MLSLocal. }
965
966 MLSLocal[1]:=solvector[1,1];
967 MLSLocal[2]:=solvector[2,1];
968 MLSLocal[3]:=solvector[3,1];
969 END;{PROCEDURE LocalAircraft_to_LocalMLSOrientation}
970
971
972
973 PROCEDURE Calculate_Uvector(rollangle,pitchangle,headingangle:DOUBLE;
974 zerovector :LOCAL;
975 VAR Uvector :LOCAL);
976
977 VAR rollmatrix,pitchmatrix,headingmatrix,helpmatrix,solmatrix,
978 vector,solvector :TNMATRIX;
979
980 {*****}
981 {-Input ; rollangle }
982 { pitchangle }
983 { headingangle }
984 { zerovector }
985 {-Output; VectorLocal }
986 { }
987 {This procedure calculates the Uvector. The Uvector is defined in a local }
988 {system in which -the x-axis is parallel to the meridians pointing eastwards,}
989 { -the y-axis is perpendicular to the x-axis pointing }
990 { northwards, }

```

```

991 {           -the x-axis is perpendicular to the xy-plane pointing into }
992 {           space. }
993 {The zerovector is the vector between the airborne MLS-antenna and a defined }
994 {point within the airplane (for example GPS-antenna or the wheels) when }
995 {having a zero roll, pitch and heading. }
996 {The Uvector describes the zerovector during change of roll, pitch and }
997 {heading. }
998 {*****}
999
1000 BEGIN
1001 {-----}
1002 {Conversion from degrees into radials. }
1003
1004 rollangle :=rollangle *pi/180;
1005 pitchangle :=pitchangle *pi/180;
1006 headingangle:=headingangle*pi/180;
1007
1008 {-----}
1009 {Create the ROLLmatrix. }
1010
1011 rollmatrix[1,1]:= 1;
1012 rollmatrix[1,2]:= 0;
1013 rollmatrix[1,3]:= 0;
1014 rollmatrix[2,1]:= 0;
1015 rollmatrix[2,2]:= cos(rollangle);
1016 rollmatrix[2,3]:= sin(rollangle);
1017 rollmatrix[3,1]:= 0;
1018 rollmatrix[3,2]:= -sin(rollangle);
1019 rollmatrix[3,3]:= cos(rollangle);
1020
1021 {-----}
1022 {Create the PITCHmatrix. }
1023
1024 pitchmatrix[1,1]:= cos(pitchangle);
1025 pitchmatrix[1,2]:= 0;
1026 pitchmatrix[1,3]:= sin(pitchangle);
1027 pitchmatrix[2,1]:= 0;
1028 pitchmatrix[2,2]:= 1;
1029 pitchmatrix[2,3]:= 0;
1030 pitchmatrix[3,1]:= -sin(pitchangle);
1031 pitchmatrix[3,2]:= 0;
1032 pitchmatrix[3,3]:= cos(pitchangle);
1033
1034 {-----}
1035 {Create the HEADINGmatrix. }
1036
1037 headingmatrix[1,1]:= cos(headingangle);
1038 headingmatrix[1,2]:= sin(headingangle);
1039 headingmatrix[1,3]:= 0;
1040 headingmatrix[2,1]:= -sin(headingangle);
1041 headingmatrix[2,2]:= cos(headingangle);
1042 headingmatrix[2,3]:= 0;
1043 headingmatrix[3,1]:= 0;
1044 headingmatrix[3,2]:= 0;
1045 headingmatrix[3,3]:= 1;
1046
1047 {-----}
1048 {Write zerovector to vector in order to use the procedure Matrix_Mult. }
1049
1050 vector[1,1]:=zerovector[1];
1051 vector[2,1]:=zerovector[2];
1052 vector[3,1]:=zerovector[3];
1053
1054 {-----}
1055 {ROLLmatrix*PITCHmatrix*HEADINGmatrix = solmatrix. }
1056
1057 Matrix_Mult(3,3,3,pitchmatrix,headingmatrix,helpmatrix);
1058 Matrix_Mult(3,3,3,rollmatrix,helpmatrix,solmatrix);
1059
1060 {-----}
1061 {solmatrix*vector = solvector. }
1062
1063 Matrix_Mult(3,3,1,solmatrix,vector,solvector);
1064
1065 {-----}
1066 {Convert the solvector to Uvector. }
1067
1068 Uvector[1]:=solvector[1,1];
1069 Uvector[2]:=solvector[2,1];
1070 Uvector[3]:=solvector[3,1];
1071 END;{PROCEDURE Calculate_Uvector}
1072
1073
1074
1075 PROCEDURE Calculate_Localcoordinates_of_Satellites(NoOfGPSequations:INTEGER;
1076                                                    SatEcef           :SATECEF;
1077                                                    VAR SatLocal       :SATLOCAL);
1078 VAR ecefvector :ECEF;
1079     localvector:LOCAL;
1080     i,j         :INTEGER;
1081
1082 {*****}
1083 {-Input ; NoOfGPSequations (=Number of used GPS-equations) }
1084 {      SatEcef (The satellitepositions in ECEF) }
1085 {-Output; SatLocal (The satellitepositions in Local Reference) }
1086 { }
1087 {This procedure transforms the satellitecoordinates in ECEF to }
1088 {satellitescoordinates in Local Reference. }
1089 {*****}
1090
1091 BEGIN
1092 {-----}
1093 {Change ECEF-origin to MLS Local Reference origin. }
1094
1095 FOR i:=1 TO NoOfGPSequations DO
1096 BEGIN
1097     FOR j:=1 TO 3 DO
1098         SatEcef[i,j]:=SatEcef[i,j]-Datumpoint_ecef[j];
1099     END;
1100

```

```

1101 {-----}
1102 {Calculate the Local coordinates of the satellites.}
1103
1104 FOR i:=1 TO NoOfGPSequations DO
1105 BEGIN
1106     FOR j:=1 TO 3 DO ecefvector[j]:=SatEcef[i,j];
1107     ECEF_to_LocalOrientation(ecefvector,Datumpoint_wgs84,North_angle,
1108                             localvector);
1109     FOR j:=1 TO 3 DO SatLocal[i,j]:=localvector[j];
1110 END;
1111 END;(PROCEDURE Calculate_Localcoordinates_of_Satellites)
1112
1113
1114
1115 PROCEDURE Create_GPSVarianceMatrix(NoOfGPSequations:INTEGER;
1116                                     GPSSeq           :GPSROW;
1117                                     VarRgps           :RANGE;
1118                                     VAR VMatrix       :TNMATRIX);
1119 VAR i,j:INTEGER;
1120
1121 {-----}
1122 {-Input ; NoOfGPSequations (= Number of used GPS-equations)}
1123 {      GPSSeq (array of used GPS-equations)}
1124 {      VarRgps (the variances of the GPS-pseudoranges)}
1125 {-Output; VMatrix (the variancematrix)}
1126 {
1127 {This procedure creates the GPSVARIANCEmatrix.}
1128 {-----}
1129
1130 BEGIN
1131 {-----}
1132 {Make all elements of the GPSVARIANCEmatrix zero.}
1133
1134 FOR i:=1 TO NoOfGPSequations DO
1135 BEGIN
1136     FOR j:=1 TO NoOfGPSequations DO Vmatrix[i,j]:=0;
1137 END;
1138
1139 {-----}
1140 {Fill in the variance(s) of the GPS-pseudorange(s).}
1141
1142 FOR i:=1 TO NoOfGPSequations DO
1143 BEGIN
1144     Vmatrix[i,i]:=VarRgps [GPSSeq[i]];
1145 END;
1146
1147 END;(PROCEDURE Create_GPSVarianceMatrix)
1148
1149
1150
1151 PROCEDURE Create_MIASVarianceMatrix(NoOfGPSequations,NoOfMLSequations:INTEGER;
1152                                     GPSSeq           :GPSROW;
1153                                     MLSeq            :MLSRROW;
1154                                     VarRdgps         :RANGE;
1155                                     VarMLS           :MLS;

```

```

1156                                     VAR VMatrix       :TNMATRIX);
1157 VAR i,j:INTEGER;
1158
1159 {-----}
1160 {-Input ; NoOfGPSequations (= Number of used GPS-equations)}
1161 {      NoOfMLSequations (= Number of used MLS-equations)}
1162 {      GPSSeq (array of used GPS-equations)}
1163 {      MLSeq (array of used MLS-equations)}
1164 {      VarRdgps (the variances of the DGPS-pseudoranges)}
1165 {      VarMLS (the variance of the azimuthangle, elevationangle and
1166 {      DME-distance)}
1167 {-Output; VMatrix (the variancematrix)}
1168 {
1169 {This procedure creates the VARIANCEmatrix.}
1170 {-----}
1171
1172 BEGIN
1173 {-----}
1174 {Make all elements of the VARIANCEmatrix zero.}
1175
1176 FOR i:=1 TO (NoOfGPSequations+NoOfMLSequations) DO
1177 BEGIN
1178     FOR j:=1 TO (NoOfGPSequations+NoOfMLSequations) DO Vmatrix[i,j]:=0;
1179 END;
1180
1181 {-----}
1182 {Fill in the variance(s) of the DGPS-pseudorange(s).}
1183
1184 FOR i:=1 TO NoOfGPSequations DO Vmatrix[i,i]:=VarRdgps [GPSSeq[i]];
1185
1186 {-----}
1187 {Fill in the variance of the azimuthangle (and/or) the elevationangle
1188 {(and/or) the DME-distance.}
1189
1190 FOR i:=1 TO NoOfMLSequations DO
1191 BEGIN
1192     Vmatrix[NoOfGPSequations+i,NoOfGPSequations+i]:=VarMLS [MLSeq[i]];
1193 END;
1194
1195 END;(PROCEDURE Create_MIASVarianceMatrix)
1196
1197
1198
1199 PROCEDURE Create_MIASweighMatrix(NoOfGPSequations,NoOfMLSequations:INTEGER;
1200                                   GPSSeq           :GPSROW;
1201                                   MLSeq            :MLSRROW;
1202                                   VarRgps         :RANGE;
1203                                   VarMLS         :MLS;
1204                                   mlsdata        :MLSBLOCK;
1205                                   VAR VMatrix       :TNMATRIX);
1206 VAR i,j:INTEGER;
1207
1208 {-----}
1209 {-Input ; NoOfGPSequations (= Number of used GPS-equations)}
1210 {      NoOfMLSequations (= Number of used MLS-equations)}

```

```

1211 (      GPSeq (array of used GPS-equations)      )
1212 (      MLSeq (array of used MLS-equations)      )
1213 (      VarRdgps (the variances of the DGPS-pseudoranges) )
1214 (      VarMLS (the variance of the azimuthangle, elevationangle and )
1215 (      DME-distance) )
1216 (-Output; VMatrix (the variancematrix) )
1217 ( )
1218 {This procedure creates the VARIANCEmatrix.}
1219 {*****}
1220
1221 BEGIN
1222 {-----}
1223 {Make all elements of the VARIANCEmatrix zero.}
1224
1225 FOR i:=1 TO (NoOfGPSequations+NoOfMLSequations) DO
1226 BEGIN
1227     FOR j:=1 TO (NoOfGPSequations+NoOfMLSequations) DO Wmatrix[i,j]:=0;
1228 END;
1229
1230 {-----}
1231 {Fill in the variance(s) of the DGPS-pseudorange(s).}
1232
1233 FOR i:=1 TO NoOfGPSequations DO Wmatrix[i,i]:=VarRgps[GPSeq[i]];
1234
1235 {-----}
1236 {Fill in the variance of the azimuthangle (and/or) the elevationangle }
1237 {(and/or) the DME-distance.}
1238
1239 FOR i:=1 TO NoOfMLSequations DO
1240 BEGIN
1241     If mlseq[i]=3
1242     Then Wmatrix[noofgpsequations+i, noofgpsequations+i]:= varmls[ mlseq[i]]
1243     Else Wmatrix[noofgpsequations+i, noofgpsequations+i]:=  sqr( mlsdata[i,3] *
1244                               sin( varmls[ mlseq[i]]));
1245 END;
1246
1247 END;{PROCEDURE Create_MIASVarianceMatrix}
1248
1249
1250
1251
1252 PROCEDURE New_GPSEstimation(NoOfGPSequations:INTEGER;
1253                             GPSeq          :GPSROW;
1254                             EstEcef        :ECEF;
1255                             SatEcef        :SATECEF;
1256                             VAR REst       :RANGE);
1257 VAR i:INTEGER;
1258
1259 {*****}
1260 {-Input ; NoOfGPSequations (=Number of used GPS-equations) }
1261 (      GPSeq (array of used GPS-equations) )
1262 (      EstEcef(the estimate of the airborne GPS-antennaposition in ECEF- )
1263 (      coordinates) )
1264 (      SatEcef (the satellitepositions in ECEF-coordinates) )
1265 (-Output; REst (the estimated ranges between the airborne GPS-antenna and )

```

```

1266 (      the satellites) )
1267 ( )
1268 {This procedure calculates the GPS-parameters needed for the coefficients of }
1269 {the GPS-matrix and the GPSvector.}
1270 {*****}
1271
1272 BEGIN
1273 {-----}
1274 {Calculate the estimated ranges between the airborne GPS-antenna and the }
1275 {satellites.}
1276
1277 FOR i:=1 TO NoOfGPSequations DO
1278 BEGIN
1279     REst[i]:=sqrt(sqr(EstEcef[1]-SatEcef[GPSeq[i],1])+
1280                   sqr(EstEcef[2]-SatEcef[GPSeq[i],2])+
1281                   sqr(EstEcef[3]-SatEcef[GPSeq[i],3]));
1282 END;
1283 END;{PROCEDURE New_GPSEstimation}
1284
1285
1286
1287 PROCEDURE New_MIASEstimation(NoOfGPSequations,NoOfMLSequations :INTEGER;
1288                             GPSeq          :GPSROW;
1289                             MLSeq         :MLSROW;
1290                             EstLocal,AziLocal,EleLocal,DMELocal:LOCAL;
1291                             SatLocal      :SATLOCAL;
1292                             VAR REst      :RANGE;
1293                             VAR MLSdata   :MLSBLOCK);
1294 VAR i:INTEGER;
1295
1296 {*****}
1297 {-Input ; NoOfGPSequations (=Number of used GPS-equations) }
1298 (      NoOfMLSequations (=Number of used MLS-equations) )
1299 (      GPSeq (array of used GPS-equations) )
1300 (      MLSeq (array of used MLS-equations) )
1301 (      EstLocal (the estimate of the airborne MLS-antennaposition in Local) )
1302 (      coordinates) )
1303 (      AziLocal,EleLocal,DMELocal(the position of the groundantennas in )
1304 (      Local coordinates) )
1305 (      SatLocal (the satellitepositions in Local coordinates) )
1306 (-Output; REst (the estimated ranges between the airborne MLS-antenna and )
1307 (      the satellites) )
1308 (      MLSdata (the MLS-data) )
1309 ( )
1310 {This procedure calculates the GPS- and MLS-parametres needed for the }
1311 {coefficients of the MIASmatrix and the MIASvector.}
1312 {*****}
1313
1314 var
1315 value :      String;
1316
1317 BEGIN
1318 {-----}
1319 {Calculate the estimated ranges between the airborne MLS-antenna and the }
1320 {satellites.}

```

```

1321
1322 FOR i:=1 TO NoOfGPSequations DO
1323 BEGIN
1324   REst[i]:=sqrt(sqr(EstLocal[1]-SatLocal[GPSeq[i],1])+
1325                 sqr(EstLocal[2]-SatLocal[GPSeq[i],2])+
1326                 sqr(EstLocal[3]-SatLocal[GPSeq[i],3]));
1327 END;
1328
1329 {-----}
1330 {Calculate the MLS-data.}
1331
1332 FOR i:=1 TO NoOfMLSequations DO
1333 BEGIN
1334   IF MLSeq[i]=1 THEN
1335     BEGIN
1336       {The estimated azimuthangle}
1337
1338       If ( sqrt(Estlocal[1] - Azilocal[1]) +
1339           sqrt(Estlocal[3] - Azilocal[3]) = 0)
1340       Then If (( estlocal[2] - azilocal[2]) >= 0)
1341           Then MLSdata[i,1]:= pi / 2
1342           Else MLSdata[i,1]:= - pi / 2
1343       Else MLSdata[i,1]:=-arctan(( EstLocal[2]-AziLocal[2])/
1344                                sqrt( sqr(EstLocal[1]-AziLocal[1])+
1345                                sqr(EstLocal[3]-AziLocal[3])));
1346
1347       {Azimuthangle accent}
1348
1349       If (( Estlocal[1] - Azilocal[1]) = 0)
1350       Then If (( Estlocal[3] - azilocal[3]) >= 0)
1351           Then MLSdata[i,2]:= pi / 2
1352           Else MLSdata[i,2]:= - pi / 2
1353       Else MLSdata[i,2]:= arctan(( EstLocal[3]-AziLocal[3])/
1354                                (EstLocal[1]-AziLocal[1]));
1355
1356       {Distance between azimuthantenna and estimated position}
1357
1358       MLSdata[i,3]:= sqrt(sqr(EstLocal[1]-AziLocal[1])+
1359                           sqr(EstLocal[2]-AziLocal[2])+
1360                           sqr(EstLocal[3]-AziLocal[3]));
1361
1362     END;
1363   IF MLSeq[i]=2 THEN
1364     BEGIN
1365       {The estimated elevationangle}
1366
1367       If ( sqrt(Estlocal[1] - Elelocal[1]) +
1368           sqrt(Estlocal[2] - Elelocal[2]) = 0)
1369       Then If (( estlocal[3] - elelocal[3]) >= 0)
1370           Then MLSdata[i,1]:= pi / 2
1371           Else MLSdata[i,1]:= - pi / 2
1372       Else MLSdata[i,1]:=-arctan(( EstLocal[3]-EleLocal[3])/
1373                                sqrt( sqr(EstLocal[1]-EleLocal[1])+
1374                                sqr(EstLocal[2]-EleLocal[2])));
1375

```

```

1376   {Elevationangle accent}
1377
1378   If (( Estlocal[1] - Elelocal[1]) = 0)
1379   Then If (( Estlocal[2] - Elelocal[2]) >= 0)
1380       Then MLSdata[i,2]:= pi / 2
1381       Else MLSdata[i,2]:= - pi / 2
1382   Else MLSdata[i,2]:= arctan(( EstLocal[2]-EleLocal[2])/
1383                             (EstLocal[1]-EleLocal[1]));
1384
1385   {Distance between the elevationantenna and the estimated position}
1386
1387   MLSdata[i,3]:= sqrt(sqr(EstLocal[1]-EleLocal[1])+
1388                       sqr(EstLocal[2]-EleLocal[2])+
1389                       sqr(EstLocal[3]-EleLocal[3]));
1390 END;
1391 IF MLSeq[i]=3 THEN
1392 BEGIN
1393   {The estimated DME-distance}
1394
1395   MLSdata[i,1]:= sqrt(sqr(EstLocal[1]-DMELocal[1])+
1396                       sqr(EstLocal[2]-DMELocal[2])+
1397                       sqr(EstLocal[3]-DMELocal[3]));
1398
1399   {DME-angle 1}
1400
1401   If (( sqrt( Estlocal[1] - dmelocal[1]) +
1402       sqrt( Estlocal[2] - dmelocal[2]) = 0)
1403   Then If (( Estlocal[3] - dmelocal[3]) >= 0)
1404       Then MLSdata[i,2]:= pi / 2
1405       Else MLSdata[i,2]:= - pi / 2
1406   Else MLSdata[i,2]:= arctan(( EstLocal[3]-DMELocal[3])/
1407                             sqrt( sqr(EstLocal[1]-DMELocal[1])+
1408                             sqr(EstLocal[2]-DMELocal[2])));
1409
1410   {DME-angle 2}
1411
1412   If (( Estlocal[1] - dmelocal[1]) = 0)
1413   Then If (( Estlocal[2] - dmelocal[2]) >= 0)
1414       Then MLSdata[i,2]:= pi / 2
1415       Else MLSdata[i,2]:= - pi / 2
1416   Else MLSdata[i,3]:= arctan(( EstLocal[2]-DMELocal[2])/
1417                             ( EstLocal[1]-DMELocal[1]));
1418 END;
1419 END;
1420
1421
1422 PROCEDURE Adjust_Measured_Pseudoranges_to_MLSantenna
1423   (NoOfGPSequations :INTEGER;
1424    GPSseq            :GPSROW;
1425    AntennaZerovector:LOCAL;
1426    EstLocal          :LOCAL;
1427    SatLocal          :SATLOCAL;
1428    REst              :RANGE;
1429    RMeas             :RANGE;
1430    VAR RAdjust       :RANGE);

```



```

1431
1432 VAR Ulocal1,
1433       Ulocal2      :LOCAL;
1434       Rdiff        :RANGE;
1435       i            :INTEGER;
1436
1437 {*****}
1438 {-Input ; NoOfGPSequations (=Number of used GPS-equations) }
1439 {      NoOfMLSequations (=Number of used MLS-equations) }
1440 {      AntennaZerovector (the vector between the airborne MLS-antenna }
1441 {                        (origin) and the airborne GPS-antenna) }
1442 {      EstLocal (the estimated position of the airborne MLS-antenna) }
1443 {      SatLocal (the satellitepositions in Local Reference) }
1444 {      REst (the distances between the estimated position and the }
1445 {            satellites) }
1446 {      RMeas (the measured pseudoranges) }
1447 {-Output; RAdjust (the adjusts pseudoranges) }
1448 { }
1449 {This procedure adjusts the measured pseudoranges (the distance between the }
1450 {satellites to the airborne GPS-antenna) to ranges from the satellites to the}
1451 {airborne MLS-antenna. }
1452 {*****}
1453
1454 BEGIN
1455 {-----}
1456 {Calculate the Uvector described in procedure Calculate_Uvector. }
1457
1458 Ulocal1[ 1]:=0;
1459 Ulocal1[ 2]:=0;
1460 Ulocal1[ 3]:=0;
1461
1462
1463
1464 If ( present.att = 1) and
1465     ( present.hdg = 1)
1466 Then
1467     Calculate_Uvector(Rollangle,Pitchangle,Headingangle,
1468                      AntennaZerovector,
1469                      Ulocal1);
1470
1471 {-----}
1472 {Convert the local aircraftcoordinatesystem in a system orientated as the }
1473 {MLS-Local Reference system. }
1474
1475 LocalAircraft_to_LocalMLSOrientation(Ulocal1,North_angle,Ulocal2);
1476
1477 {-----}
1478 {Calculate the differences in distance between the estimated position }
1479 {of the airborne GPS-antenna and the satellitepositions and between }
1480 {the estimated position of the airborne MLS-antenna and the }
1481 {satellitepositions }
1482
1483 FOR i:=1 TO NoOfGPSequations DO
1484 BEGIN
1485     Rdiff[i]:=sqrt(sqr(EstLocal[1]+Ulocal2[1]-SatLocal[GPSeq[i],1])+
1486
1487
1488     END;
1489
1490 {-----}
1491 {Subtract these differences from the measured pseudoranges in order }
1492 {to get a "measured" range between the airborne MLS-antenna and the }
1493 {satellites }
1494
1495 FOR i:=1 TO NoOfGPSequations DO
1496 BEGIN
1497     RAdjust[GPSeq[i]]:=RMeas[GPSeq[i]]-Rdiff[i];
1498 END;
1499
1500 END;{PROCEDURE Adjust_Measured_Pseudoranges_to_MLSantenna}
1501
1502
1503
1504 PROCEDURE Calculate_GPSmatrix(NoOfGPSequations:INTEGER;
1505                               GPSeq      :GPSROW;
1506                               EstEcef    :ECEF;
1507                               REst       :RANGE;
1508                               SatEcef    :SATECEF;
1509                               VAR Matrix :TNMATRIX);
1510 VAR i:INTEGER;
1511
1512 {*****}
1513 {-Input ; NoOfGPSequations (=Number of used GPS-equations) }
1514 {      GPSeq (array of used GPS-equations) }
1515 {      EstEcef (the estimated position of the airborne GPS-antenna in }
1516 {            ECEF-coordinates) }
1517 {      REst (the distance between the estimated position and the }
1518 {            satellites) }
1519 {      SatEcef (the satellitepositions in ECEF-coordinates) }
1520 {-Output; Matrix (GPSmatrix) }
1521 { }
1522 {This procedure creates the GPS-matrix using the parametres calculated by }
1523 {the procedure New_GPSEstimation. }
1524 {*****}
1525
1526 BEGIN
1527 {-----}
1528 {The GPS-equations. }
1529
1530 FOR i:=1 TO NoOfGPSequations DO
1531 BEGIN
1532     Matrix[i,1]:=(EstEcef[1]-SatEcef[GPSeq[i],1])/REst[i];
1533     Matrix[i,2]:=(EstEcef[2]-SatEcef[GPSeq[i],2])/REst[i];
1534     Matrix[i,3]:=(EstEcef[3]-SatEcef[GPSeq[i],3])/REst[i];
1535     Matrix[i,4]:=-1;
1536 END;
1537 END;{PROCEDURE Calculate_GPSmatrix}
1538
1539
1540

```

```

1541 PROCEDURE Calculate_MIASmatrix(NoOfGPSequations,NoOfMLSequations:INTEGER;
1542                                GPSeq          :GPSROW;
1543                                MLSeq          :MLSROW;
1544                                EstLocal      :LOCAL;
1545                                REst         :RANGE;
1546                                SatLocal     :SATLOCAL;
1547                                MLSdata     :MLSBLOCK;
1548                                VAR Matrix   :TNMATRIX);
1549 VAR i:INTEGER;
1550
1551 {*****}
1552 {-Input ; NoOfGPSequations (=Number of used GPS-equations) }
1553 { NoOfMLSequations (=Number of used MLS-equations) }
1554 { GPSeq (array of used GPS-equations) }
1555 { MLSeq (array of used MLS-equations) }
1556 { EstLocal (the estimated position of the airborne MLS-antenna in }
1557 { Local Reference coordinates) }
1558 { REst (the distance between the estimated position and the }
1559 { satellites) }
1560 { SatLocal (the satellitepositions in Local Reference coordinates) }
1561 { MLSdata (estimated MLS-data+extra MLS-data) }
1562 {-Output; Matrix (MIASmatrix) }
1563 {
1564 {This procedure creates the MIASmatrix using the parametres calculated by }
1565 {the procedure New_Estimation. }
1566 {*****}
1567 BEGIN
1568 {-----}
1569 {The GPS-equations. }
1570
1571 FOR i:=1 TO NoOfGPSequations DO
1572 BEGIN
1573 Matrix[i,1]:=(EstLocal[1]-SatLocal[GPSeq[i],1])/REst[i];
1574 Matrix[i,2]:=(EstLocal[2]-SatLocal[GPSeq[i],2])/REst[i];
1575 Matrix[i,3]:=(EstLocal[3]-SatLocal[GPSeq[i],3])/REst[i];
1576 Matrix[i,4]:=-1;
1577 END;
1578
1579 {-----}
1580 {The MLS-equations. }
1581
1582 FOR i:=1 TO NoOfMLSequations DO
1583 BEGIN
1584 IF MLSeq[i]=1 THEN {The azimuthequation}
1585 BEGIN
1586 IF (MLSdata[i,3] = 0)
1587 THEN Begin
1588 IF ( -sin(MLSdata[i,1]) * cos(MLSdata[i,2]) >= 0)
1589 THEN if ( -sin(MLSdata[i,1]) * cos(MLSdata[i,2]) = 0)
1590 THEN Matrix[NoOfGPSequations+i,1]:=1
1591 ELSE Matrix[NoOfGPSequations+i,1]:=0
1592 ELSE if ( -sin(MLSdata[i,1]) * cos(MLSdata[i,2]) = 0)
1593 THEN Matrix[NoOfGPSequations+i,1]:=-1
1594 ELSE Matrix[NoOfGPSequations+i,1]:=0;

```

```

1596
1597 IF ( -cos(MLSdata[i,1]) >=0)
1598 THEN If ( -cos(MLSdata[i,1]) =0)
1599 THEN Matrix[NoOfGPSequations+i,2]:= 1
1600 ELSE Matrix[NoOfGPSequations+i,2]:= 0
1601 ELSE If ( -cos(MLSdata[i,1]) =0)
1602 THEN Matrix[NoOfGPSequations+i,2]:= -1
1603 ELSE Matrix[NoOfGPSequations+i,2]:= 0;
1604
1605 IF ( -sin(MLSdata[i,1]) * sin(MLSdata[i,2]) >= 0)
1606 THEN If ( -sin(MLSdata[i,1]) * sin(MLSdata[i,2]) = 0)
1607 THEN Matrix[NoOfGPSequations+i,3]:= 1
1608 ELSE Matrix[NoOfGPSequations+i,3]:= 0
1609 ELSE If ( -sin(MLSdata[i,1]) * sin(MLSdata[i,2]) = 0)
1610 THEN Matrix[NoOfGPSequations+i,3]:= -1
1611 ELSE Matrix[NoOfGPSequations+i,3]:= 0;
1612
1613 End
1614 ELSE Begin
1615 Matrix[NoOfGPSequations+i,1]:= -sin(MLSdata[i,1])*cos(MLSdata[i,2])/
1616 MLSdata[i,3];
1617 Matrix[NoOfGPSequations+i,2]:= -cos(MLSdata[i,1])/MLSdata[i,3];
1618 Matrix[NoOfGPSequations+i,3]:= -sin(MLSdata[i,1])*sin(MLSdata[i,2])/
1619 MLSdata[i,3];
1620 End;
1621 Matrix[NoOfGPSequations+i,4]:= 0;
1622 END;
1623 IF MLSeq[i]=2 THEN {The elevationequation}
1624 BEGIN
1625 IF (MLSdata[i,3] = 0)
1626 THEN Begin
1627 IF ( -sin(MLSdata[i,1]) * cos(MLSdata[i,2]) >= 0)
1628 THEN if ( -sin(MLSdata[i,1]) * cos(MLSdata[i,2]) = 0)
1629 THEN Matrix[NoOfGPSequations+i,1]:=1
1630 ELSE Matrix[NoOfGPSequations+i,1]:=0
1631 ELSE if ( -sin(MLSdata[i,1]) * cos(MLSdata[i,2]) = 0)
1632 THEN Matrix[NoOfGPSequations+i,1]:=-1
1633 ELSE Matrix[NoOfGPSequations+i,1]:=0;
1634
1635 IF ( sin(MLSdata[i,1]) * sin(MLSdata[i,2]) >= 0)
1636 THEN If ( sin(MLSdata[i,1]) * sin(MLSdata[i,2]) = 0)
1637 THEN Matrix[NoOfGPSequations+i,3]:= 1
1638 ELSE Matrix[NoOfGPSequations+i,3]:= 0
1639 ELSE If ( sin(MLSdata[i,1]) * sin(MLSdata[i,2]) = 0)
1640 THEN Matrix[NoOfGPSequations+i,3]:= -1
1641 ELSE Matrix[NoOfGPSequations+i,3]:= 0;
1642
1643 IF ( cos(MLSdata[i,1]) >=0)
1644 THEN If ( cos(MLSdata[i,1]) =0)
1645 THEN Matrix[NoOfGPSequations+i,2]:= 1
1646 ELSE Matrix[NoOfGPSequations+i,2]:= 0
1647 ELSE If ( cos(MLSdata[i,1]) =0)
1648 THEN Matrix[NoOfGPSequations+i,2]:= -1
1649 ELSE Matrix[NoOfGPSequations+i,2]:= 0;
1650

```

```

1651                                     MLSeq                               :MLSROW;
1652                                     REst,RAdjust                          :RANGE;
1653     Else Begin                                     MLSdata                     :MLSBLOCK;
1654         Matrix[NoOfGPSequations+i,1]:= sin(MLSdata[i,1])*sin(MLSdata[i,2])/ 1708                                     MLSMeas                          :MLS;
1655         MLSdata[i,3];                                     VAR Vector                          :TNMATRIX);
1656         Matrix[NoOfGPSequations+i,2]:= cos(MLSdata[i,1])/MLSdata[i,3]; 1711 VAR i:INTEGER;
1657         Matrix[NoOfGPSequations+i,3]:= sin(MLSdata[i,1])*sin(MLSdata[i,2])/ 1712
1658         MLSdata[i,3];                                     1713 {*****}
1659     End;                                     1714 {-Input ; NoOfGPSequations (=Number of used GPS-equations) }
1660     Matrix[NoOfGPSequations+i,4]:= 0; 1715 {      NoOfMLSequations (=Number of used MLS-equations) }
1661 END; 1716 {      GPSeq (array of used GPS-equations) }
1662 IF MLSeq[i]=3 THEN {The DME-distance-equation} 1717 {      MLSeq (array of used MLS-euations) }
1663 BEGIN 1718 {      REst (the distance between the estimated position and the
1664     Matrix[NoOfGPSequations+i,1]:= cos(MLSdata[i,2])*cos(MLSdata[i,3]); 1719 {      satellites) }
1665     Matrix[NoOfGPSequations+i,2]:= cos(MLSdata[i,2])*sin(MLSdata[i,3]); 1720 {      RAdjust (the distance between the airborne MLS-antenna and the
1666     Matrix[NoOfGPSequations+i,3]:= sin(MLSdata[i,2]); 1721 {      satellites) }
1667     Matrix[NoOfGPSequations+i,4]:= 0; 1722 {      MLSdata (estimated MLS-data) }
1668 END; 1723 {      MLSMeas (measured MLS-data) }
1669 END; 1724 {-Output; Vector (the MIASvector) }
1670 END;{PROCEDURE Calculate_MIASmatrix} 1725 {
1671 1726 {This procedure creates the MIASvector using the parametres calculated by the}
1672 1727 {procedure New_Estimation and the measured GPS- and MLS-parametres. }
1673 1728 {*****}
1674 PROCEDURE Calculate_GPSvector(NoOfGPSequations:INTEGER; 1729
1675     GPSeq :GPSROW; 1730 BEGIN
1676     REst,RMeas :RANGE; 1731 {-----}
1677     VAR Vector :TNMATRIX); 1732 {The GPS-parametres. }
1678 VAR i:INTEGER; 1733
1679 1734 FOR i:=1 TO NoOfGPSequations DO
1680 {*****} 1735 BEGIN
1681 {-Input ; NoOfGPSequations (=Number of used GPS-equations) } 1736     Vector[i,1]:=RAdjust[GPSeq[i]]-REst[i];
1682 {      GPSeq (array of used GPS-equations) } 1737 END;
1683 {      REst (the distance between the estimated position of the airborne } 1738
1684 {      GPS-antenna and the satellites) } 1739 {-----}
1685 {      RMeas (the pseudoranges) } 1740 {The MLS-parametres. }
1686 {-Output; Vector (the GPSvector) } 1741
1687 { } 1742 FOR i:=1 TO NoOfMLSequations DO
1688 {This procedure creates the GPSvector using the parametres calculated by the } 1743 BEGIN
1689 {procedure New_GPSEstimation and the measured pseudoranges. } 1744     Vector[NoOfGPSequations+i,1]:=MLSmeas[MLSeq[i]]-MLSdata[i,1];
1690 {*****} 1745 END;
1691 1746 END;{PROCEDURE Calculate_MIASvector}
1692 BEGIN 1747
1693 {-----} 1748
1694 {The GPS-parametres. } 1749
1695 1750 PROCEDURE Calc_Weighed_LeastSqMatrix(RowTotal :INTEGER;
1696 FOR i:=1 TO NoOfGPSequations DO 1751     A,
1697 BEGIN 1752     V :TNMATRIX;
1698     Vector[i,1]:=RMeas[GPSeq[i]]-REst[i]; 1753     VAR invAtHA_AtH:TNMATRIX;
1699 END; 1754     Error :BYTE);
1700 END;{PROCEDURE Calculate_GPSvector} 1755
1701 1756 VAR H, {The inverse of the variancematrix V}
1702     At, {The transpose of the matrix A}
1703     AtH, {The transpose of A multiplied with H}
1704 PROCEDURE Calculate_MIASvector(NoOfGPSequations,NoOfMLSequations:INTEGER; 1759     AtHA, {The matrix AtH multiplied with A}
1705     GPSeq :GPSROW; 1760     invAtHA {The inverse of At*HA}

```

```

1761      :TNMATRIX;
1762      i      :INTEGER;
1763
1764 {*****}
1765 {-Input ; RowTotal (The total amount of rows) }
1766 {      A (= The MIASmatrix) }
1767 {      V (= The VARIANCEmatrix) }
1768 {-Output; invAtHA_AtH (= The weighed Least-SquareMatrix) }
1769 {
1770 {This procedure calculates the weighed Least-squareMatrix which is the matrix}
1771 {inverse(At*H*A)*At*H in which At is the transpose of A and H is the inverse }
1772 {of the VARIANCEmatrix V. }
1773 {*****}
1774
1775 var
1776 value      : string;
1777 gdop       : Double;
1778
1779 BEGIN
1780   Transpose(RowTotal,4,A,At);      {Solution =      At      }
1781   Inverse(RowTotal,V,H,Error);      {Solution =      H      }
1782   Matrix_Mult(4,RowTotal,RowTotal,At,H,AtH); {Solution =      At*H      }
1783   Matrix_Mult(4,RowTotal,4,AtH,A,AtHA); {Solution =      AtH*A      }
1784   Inverse(4,AtHA,invAtHA,Error);      {Solution = inverse(AtHA) }
1785
1786   Matrix_Mult(4,4,RowTotal,invAtHA,AtH,invAtHA_AtH);
1787   {Solution = }
1788   {inverse(AtHA)*AtH }
1789 END;{PROCEDURE Calc_Weighed_LeastSqMatrix}
1790
1791
1792
1793 PROCEDURE Add_GPSPositionvector(EstEcef      :ECEF;
1794                                GPSPositionZeroVector:LOCAL;
1795                                VAR FixEcef      :ECEF);
1796 VAR Ulocal :LOCAL;
1797   EstWgs84:WGS84;
1798   Uecef   :ECEF;
1799   i       :INTEGER;
1800
1801 {*****}
1802 {-Input ; EstEcef (the final estimated position of the airborne GPS-antenna )
1803 {      in ECEF) }
1804 {      GPSPositionZeroVector (the vector between the airborne GPS-antenna )
1805 {      (origin) and a certain defined point within )
1806 {      the airplane when roll, pitch and heading )
1807 {      are zero) }
1808 {      FixEcef (the position of a certain defined point within the )
1809 {      airplane in Ecef) }
1810 {This procedure adds the positionvector to the calculated position of the }
1811 {airborne GPS-antenna. }
1812 {*****}
1813
1814 BEGIN
1815 {-----}
1816 {Calculate the Uvector out of the GPSPosition_Zerovector by using the }
1817 {roll-, pitch- and headingangle. }
1818
1819 Ulocal[ 1]:=0;
1820 Ulocal[ 2]:=0;
1821 Ulocal[ 3]:=0;
1822
1823 If ( present.att = 1) and
1824   ( present.hdg = 1)
1825 Then
1826   Calculate_Uvector(Rollangle,Pitchangle,Headingangle,
1827                   GPSPosition_Zerovector,
1828                   Ulocal);
1829
1830 {-----}
1831 {Convert the estimated ECEF-position to WGS84 in order to use procedure }
1832 {Local_to_ECEFOrientation. }
1833
1834 ECEF_to_WGS84(EstEcef,EstWgs84);
1835
1836 {-----}
1837 {The vector Ulocal converted to ECEF orientation. }
1838
1839 Local_to_ECEFOrientation(Ulocal,EstWgs84,0,Uecef);
1840
1841 {-----}
1842 {Add Uecef to EstEcef to get the final positionfix. }
1843
1844 FOR i:=1 TO 3 DO FixEcef[i]:=EstEcef[i]+Uecef[i];
1845
1846 END;{PROCEDURE Add_GPSPositionvector}
1847
1848
1849
1850 PROCEDURE Add_MIASPositionvector(EstLocal,
1851                                  MIASPositionZeroVector:LOCAL;
1852                                  VAR FixLocal          :LOCAL);
1853 VAR Ulocal1,
1854     Ulocal2 :LOCAL;
1855     i       :INTEGER;
1856
1857 {*****}
1858 {-Input ; EstLocal (the final estimated position of the airborne MLS-antenna )
1859 {      in MLS Local Reference) }
1860 {      MIASPositionZeroVector (the vector between the airborne MLS-antenna )
1861 {      (origin) and a certain defined point within the )
1862 {      airplane when roll, pitch and heading are zero)) }
1863 {      FixLocal (the position of a certain defined point within the )
1864 {      airplane in Local Reference) }
1865 {This procedure adds the positionvector to the calculated position of the }
1866 {airborne MLS-antenna. }
1867 {*****}
1868
1869 BEGIN
1870 {-----}

```

```

1871 {Calculate the Uvector out of the Position_Zerovector by using the roll-, }
1872 {pitch- and headingangle. }
1873
1874 Ulocal1[ 1]:=0;
1875 Ulocal1[ 2]:=0;
1876 Ulocal1[ 3]:=0;
1877
1878 If ( present.att = 1) and
1879 ( present.hdg = 1)
1880 Then
1881 Calculate_Uvector(Rollangle,Pitchangle,Headingangle,
1882 MIASPosition_Zerovector,
1883 Ulocal1);
1884
1885 {-----}
1886 {Convert the Uvector to MLS Local Reference-orientation. }
1887
1888 LocalAircraft_to_LocalMLSOrientation(Ulocal1,North_angle,Ulocal2);
1889
1890 {-----}
1891 {Add Ulocal2 to EstLocal to get the final positionfix. }
1892
1893 FOR i:=1 TO 3 DO FixLocal[i]:=EstLocal[i]+Ulocal2[i];
1894
1895 END;{PROCEDURE Add_MIASPositionvector}
1896
1897
1898
1899 PROCEDURE Calculate_ECEFVariance(WLSquareMatrix,
1900 VarMatrix :TNMATRIX;
1901 VAR VarianceEcef :ECEF);
1902 VAR TransWLSqMatrix,
1903 Matrix,SolMatrix:TNMATRIX;
1904 i :INTEGER;
1905
1906 {*****}
1907 {-Input ; WLSquareMatrix (The weighed Least-Squarematrix) }
1908 { VarMatrix (The VARIANCEmatrix) }
1909 {-Output; VarianceEcef (The variance in the ECEF-position) }
1910 {
1911 {This procedure calculates the variance in the MLS ECEF-position. }
1912 {The variance is given by the diagonalelements of the matrix }
1913 {WLSquareMatrix*VarMatrix*TransWLSqMatrix. }
1914 {*****}
1915
1916 BEGIN
1917 {-----}
1918 {Calculate the transpose of the WLSquareMatrix. }
1919
1920 Transpose(4,TotalOfEquations,WLSquareMatrix,TransWLSqMatrix);
1921
1922 {-----}
1923 {WLSquareMatrix*VarMatrix*TransWLSqMatrx. }
1924
1925 Matrix_Mult(TotalOfEquations,TotalOfEquations,4,VarMatrix,TransWLSqMatrix,
1926 Matrix);
1927 Matrix_Mult(4,TotalOfEquations,4,WLSquareMatrix,Matrix,SolMatrix);
1928
1929 {-----}
1930 {The variance in ECEF-position is given by the diagonalelements of the }
1931 {SolMatrix. }
1932
1933 FOR i:=1 TO 3 DO VarianceEcef[i]:=SolMatrix[i,i];
1934
1935 END;{PROCEDURE Calculate_ECEFVariance}
1936
1937
1938
1939 PROCEDURE Calculate_LOCALVariance(WLSquareMatrix,
1940 VarMatrix :TNMATRIX;
1941 VAR VarianceLocal :LOCAL);
1942 VAR TransWLSqMatrix,
1943 Matrix,SolMatrix:TNMATRIX;
1944 i :INTEGER;
1945
1946 {*****}
1947 {-Input ; WLSquareMatrix (The weighed Least-Squarematrix) }
1948 { VarMatrix (The VARIANCEmatrix) }
1949 {-Output; VarianceLocal (The variance in the Local Reference position) }
1950 {
1951 {This procedure calculates the variance in the MLS Local Reference position. }
1952 {The variance is given by the diagonalelements of the matrix }
1953 {WLSquareMatrix*VarMatrix*TransWLSqMatrix. }
1954 {*****}
1955
1956 BEGIN
1957 {-----}
1958 {Calculate the transpose of the WLSquareMatrix. }
1959
1960 Transpose(4,TotalOfEquations,WLSquareMatrix,TransWLSqMatrix);
1961
1962 {-----}
1963 {WLSquareMatrix*VarMatrix*TransWLSqMatrx. }
1964
1965 Matrix_Mult(TotalOfEquations,TotalOfEquations,4,VarMatrix,TransWLSqMatrix,
1966 Matrix);
1967 Matrix_Mult(4,TotalOfEquations,4,WLSquareMatrix,Matrix,SolMatrix);
1968
1969 {-----}
1970 {The variance in MLS Local Reference position is given by the }
1971 {diagonalelements of the SolMatrix. }
1972
1973 FOR i:=1 TO 3 DO VarianceLocal[i]:=SolMatrix[i,i];
1974
1975 END;{PROCEDURE Calculate_LOCALVariance}
1976
1977
1978
1979 PROCEDURE Calculate_Position(NoOfGPSequations,
1980 gps1,gps2,gps3,gps4,gps5,gps6,gps7,

```

```

1981      NoOfMLSequations,
1982      mls1,mls2,mls3      :INTEGER);
1983  VAR i      :INTEGER;
1984      firsttime:BOOLEAN;
1985
1986
1987 {*****}
1988 {-Input ; NoOfGPSequations (=Number of used GPS-equations) }
1989 {      gpsx (the GPS-equations used in the algorithm) }
1990 {      NoOfMLSequations (=Number of used MLS-equations) }
1991 {      mlsx (the MLS-equations used in the algorithm) }
1992 { }
1993 {This procedure calculates and prints the positionfix in ECEF, WGS84 (and }
1994 {Local Reference) using a (sub)set of equations defined in GPS_eq en MLS_eq. }
1995 {*****}
1996
1997
1998 BEGIN
1999 {-----}
2000 {Define the global variables NumberOfGPSequations and NumberOfMLSequations }
2001 {and calculate the total amount of used equations. }
2002
2003   NumberOfGPSequations:=NoOfGPSequations;
2004   NumberOfMLSequations:=NoOfMLSequations;
2005
2006   TotalOfEquations:=NumberOfGPSequations+NumberOfMLSequations;
2007
2008 {-----}
2009 {Fill in the GPS-equations used in the algorithm. }
2010
2011   GPS_eq[1]:=gps1;
2012   GPS_eq[2]:=gps2;
2013   GPS_eq[3]:=gps3;
2014   GPS_eq[4]:=gps4;
2015   GPS_eq[5]:=gps5;
2016   GPS_eq[6]:=gps6;
2017   GPS_eq[7]:=gps7;
2018
2019 {-----}
2020 {Check GPSavailable and MLS available. }
2021
2022   GPSavailable:=(NumberOfGPSequations > 0);
2023   MLSavailable:=(NumberOfMLSequations > 0);
2024
2025   IF ((MLSavailable) AND (GPSavailable) AND (TotalOfEquations >= 4))
2026   OR ((MLSavailable) AND (NumberOfMLSequations = 3)) THEN
2027   BEGIN
2028     {-----}
2029     {Fill in the MLS-equations used in the algorithm. }
2030
2031     MLS_eq[1]:=mls1;
2032     MLS_eq[2]:=mls2;
2033     MLS_eq[3]:=mls3;
2034
2035     {-----}
2036
2037     {Calculate the local coordinates of the satellites. }
2038
2039     Calculate_Localcoordinates_of_Satellites(NumberOfGPSequations,
2040                                             Sat_ecef,Sat_local);
2041
2042     {-----}
2043     {Create the MIASVARIANCEmatrix. }
2044
2045     Create_MIASVarianceMatrix(NumberOfGPSequations,NumberOfMLSequations,
2046                               GPS_eq,MLS_eq,
2047                               Var_Rdgps,Var_MLS,MIASVARIANCEmatrix);
2048
2049     iter:= 0;
2050     REPEAT
2051     BEGIN
2052       {-----}
2053       {Make a new estimation of the ranges and angles of MIAS. }
2054
2055       New_MIASEstimation(NumberOfGPSequations,NumberOfMLSequations,
2056                           GPS_eq,MLS_eq,
2057                           Est_local,Azi_local,Ele_local,DME_local,
2058                           Sat_local,
2059                           R_est,MLS_data);
2060
2061       Create_miasweighmatrix( numberofgpsequations, numberofmlsequations,
2062                               gps_eq, mls_eq,
2063                               var_rdgps, var_mls, mls_data, miasweighmatrix);
2064
2065       {-----}
2066       {Convert the measured pseudoranges to "measured" ranges between the }
2067       {the airborne MLS-antenna and the satellites. }
2068
2069       Adjust_Measured_Pseudoranges_to_MLSantenna(NumberOfGPSequations,
2070                                                   GPS_eq,
2071                                                   Antenna_Zerovector,
2072                                                   Est_local,Sat_local,R_est,
2073                                                   R_meas,R_adjust);
2074
2075       {-----}
2076       {Fill in the coefficients of the MIASmatrix. }
2077
2078       Calculate_MIASmatrix(NumberOfGPSequations,NumberOfMLSequations,
2079                             GPS_eq,MLS_eq,
2080                             Est_local,R_est,Sat_local,MLS_data,MIASmatrix);
2081
2082       {-----}
2083       {Fill in the coefficients of the MIASvector. }
2084
2085       Calculate_MIASvector(NumberOfGPSequations,NumberOfMLSequations,
2086                             GPS_eq,MLS_eq,
2087                             R_est,R_adjust,MLS_data,MLS_meas,MIASvector);
2088
2089       {-----}
2090       {Calculate the weighed Least-SquareMatrix of the MIASmatrix. }
2091
2092       Calc_Weighed_LeastSqMatrix(TotalOfEquations,MIASmatrix,

```

2091	MIASVARIANCEmatrix,WLSqMatrix,Error);	2146	Sat_ecef,
2092		2147	R_est);
2093	{-----}	2148	
2094	{Multiply the weighed Least-SquareMatrix and the MIASvector. }	2149	
2095		2150	{-----}
2096	Matrix_Mult(4,TotalOfEquations,1,WLSqMatrix,MIASvector,DeltaVector);	2151	{Fill in the coefficients of the GPSmatrix. }
2097		2152	
2098	{-----}	2153	Calculate_GPSmatrix(NumberOfGPSequations,
2099	{Update the estimated position of the airborne MLS-antenna. }	2154	GPS_eq,
2100		2155	Est_ecef,R_est,Sat_ecef,GPSmatrix);
2101	FOR i:=1 TO 4 DO Est_local[i]:=Est_local[i]+DeltaVector[i,1];	2156	
2102		2157	{-----}
2103	{-----}	2158	{Fill in the coefficients of the GPSvector. }
2104	Inc( iter);	2159	
2105	{-----}	2160	Calculate_GPSvector(NumberOfGPSequations,
2106	END;	2161	GPS_eq,
2107	UNTIL ((abs(DeltaVector[1,1]) <= convergence_error) AND	2162	R_est,R_meas,GPSvector);
2108	(abs(DeltaVector[2,1]) <= convergence_error) AND	2163	
2109	(abs(DeltaVector[3,1]) <= convergence_error) AND	2164	{-----}
2110	(abs(DeltaVector[4,1]) <= convergence_error)) Or	2165	{Calculate the weighed Least-SquareMatrix of the GPSmatrix. }
2111	( iter >= maxiter);	2166	
2112		2167	Calc_Weighed_LeastSqMatrix(NumberOfGPSequations,GPSmatrix,
2113	{accuracy of 1E-0004 metres in}	2168	GPSVARIANCEmatrix,WLSqMatrix,Error);
2114	{x-, y- and z-direction }	2169	
2115	{-----}	2170	{-----}
2116	{Calculate the variance in the LOCAL-position. }	2171	{Multiply the weighed Least-SquareMatrix and the GPSvector. }
2117		2172	
2118	Calculate_LOCALVariance(WLSqMatrix,MIASVARIANCEmatrix,Variance_local);	2173	
2119		2174	Matrix_Mult(4,NumberOfGPSequations,1,WLSqMatrix,GPSvector,
2120	{-----}	2175	DeltaVector);
2121	{Add MIASPositionvector to get a final positionfix. }	2176	
2122		2177	{-----}
2123	Add_MIASPositionvector(Est_local,MIASPosition_ZeroVector,Fix_local);	2178	{Update the estimated position of the airborne GPS-antenna. }
2124	END	2179	
2125	ELSE	2180	
2126	BEGIN	2181	{-----}
2127	IF (GPSavailable) AND (NumberOfGPSequations >= 4) THEN	2182	Inc( iter);
2128	BEGIN	2183	{-----}
2129	{-----}	2184	END;
2130	{Create the GPSVARIANCEmatrix. }	2185	UNTIL ((abs(DeltaVector[1,1]) <= convergence_error) AND
2131		2186	(abs(DeltaVector[2,1]) <= convergence_error) AND
2132	Create_GPSVarianceMatrix(NumberOfGPSequations,	2187	(abs(DeltaVector[3,1]) <= convergence_error) AND
2133	GPS_eq,	2188	(abs(DeltaVector[4,1]) <= convergence_error)) Or
2134	Var_Rgps,GPSVARIANCEmatrix);	2189	( iter >= maxiter);
2135		2190	
2136	iter:= 0;	2191	{accuracy of 1E-0004 metres}
2137	REPEAT	2192	{in x-, y- and z-direction }
2138	BEGIN	2193	
2139	{-----}	2194	{-----}
2140	{Make a new estimation of the ranges towards the airborne }	2195	{Calculate the variance in the ECEF-position. }
2141	{GPS-antenna. }	2196	
2142		2197	Calculate_ECEFFariance(WLSqMatrix,GPSVARIANCEmatrix,Variance_ecef);
2143	New_GPSEstimation(NumberOfGPSequations,	2198	
2144	GPS_eq,	2199	{-----}
2145	Est_ecef,	2200	{Add GPSPositionvector to get a final positionfix. }

```

2201      Add_GPSPositionvector(Est_ecef,GPSPosition_Zerovector,Fix_ecef);
2202      END
2203      ELSE
2204      BEGIN
2205      {      writeln('Too few equations to calculate position!');}
2206      END;
2207      END;
2208      END;{PROCEDURE Calculate_Position}
2209
2210
2211
2212      PROCEDURE Set_Measured_Values;
2213
2214      {*****}
2215      {This procedure sets the parametres needed to solve the MIAS-position. }
2216      {*****}
2217
2218      BEGIN
2219      {The variances of the GPS-pseudoranges. }
2220
2221      Var_Rgps[1]:= 30 ; {metres}
2222      Var_Rgps[2]:= 30 ;
2223      Var_Rgps[3]:= 30 ;
2224      Var_Rgps[4]:= 30 ;
2225      Var_Rgps[5]:= 30 ;
2226      Var_Rgps[6]:= 30 ;
2227      Var_Rgps[7]:= 30 ;
2228
2229      {-----}
2230      {The variances of the DGPS-pseudoranges and the azimuthangle, elevationangle }
2231      {and DME-distance. }
2232
2233      Var_Rdgps[1]:= 5 ; {metres}
2234      Var_Rdgps[2]:= 5 ;
2235      Var_Rdgps[3]:= 5 ;
2236      Var_Rdgps[4]:= 5 ;
2237      Var_Rdgps[5]:= 5 ;
2238      Var_Rdgps[6]:= 5 ;
2239      Var_Rdgps[7]:= 5 ;
2240
2241      Var_MLS[1]:= 1.7453E-0003; {0.1 degrees}
2242      Var_MLS[2]:= 1.7453E-0003; {0.1 degrees}
2243      Var_MLS[3]:= 12 ; {metres}
2244
2245      END;{PROCEDURE Set_Measured_Values}
2246
2247
2248
2249      PROCEDURE MRInterface(alldata:alldatatype);
2250
2251      VAR i:INTEGER;
2252
2253      {*****}
2254      {-Input ; alldata (all GPS- and MLS-data written in datatypes by Marco Meijer}
2255      {-Output; the same data written in datatypes used by René van Leeuwen. }

```

```

2256 {
2257 {This procedure acts as a interface between the program written by Marco }
2258 {Meijer and the program written by René van Leeuwen. }
2259 {*****}
2260
2261      BEGIN
2262      WITH alldata DO
2263      BEGIN
2264      WITH GPS DO
2265      BEGIN
2266      NumberOfGPSequations:=0;
2267      gps1:= 0;
2268      gps2:= 0;
2269      gps3:= 0;
2270      gps4:= 0;
2271      gps5:= 0;
2272      gps6:= 0;
2273      gps7:= 0;
2274
2275      FOR i:=1 TO 32 DO
2276      BEGIN
2277      IF (present = 1) And (Not flag) And (Not prn[i].flag) THEN
2278      BEGIN
2279      NumberOfGPSequations:=NumberOfGPSequations+1;
2280      Case NumberOfGPSequations OF
2281      1: gps1:=1;
2282      2: gps2:=2;
2283      3: gps3:=3;
2284      4: gps4:=4;
2285      5: gps5:=5;
2286      6: gps6:=6;
2287      7: gps7:=7;
2288      End;
2289
2290      R_meas[NumberOfGPSequations]:=prn[i].pr;
2291
2292      Sat_ecef[NumberOfGPSequations,1]:=prn[i].position.x;
2293      Sat_ecef[NumberOfGPSequations,2]:=prn[i].position.y;
2294      Sat_ecef[NumberOfGPSequations,3]:=prn[i].position.z;
2295      END;
2296      END;
2297      END;
2298      WITH MLS DO
2299      BEGIN
2300      mls1:=0;
2301      mls2:=0;
2302      mls3:=0;
2303      numberofMLSequations:= 0;
2304      IF (present = 1) And (Not flag)
2305      THEN
2306      BEGIN
2307      IF Not AZangle_flag THEN
2308      BEGIN
2309      Azi_local[1]:=Azpos.x; Azi_local[2]:=Azpos.y; Azi_local[3]:=Azpos.z;
2310      MLS_meas[1]:=AZangle;

```



```

2311     NumberOfMLSequations:=NumberOfMLSequations+1;
2312     mls1:=1;
2313 END;
2314 IF Not EAngle_flag THEN
2315 BEGIN
2316     Ele_local[1]:=Elpos.x; Ele_local[2]:=Elpos.y; Ele_local[3]:=Elpos.z;
2317     MLS_meas[2]:=EAngle;
2318     NumberOfMLSequations:=NumberOfMLSequations+1;
2319     mls2:=2;
2320 END;
2321 IF Not DME_flag THEN
2322 BEGIN
2323     DME_local[1]:=DMEpos.x; DME_local[2]:=DMEpos.y; DME_local[3]:=DMEpos.z;
2324     MLS_meas[3]:=DMErange;
2325     NumberOfMLSequations:=NumberOfMLSequations+1;
2326     mls3:=3;
2327 END;
2328 North_angle:= Runwayhdg + 90;
2329 If North_angle >= 360
2330 Then North_angle:= North_angle - 360;
2331
2332     If North_angle <= 0
2333     Then North_angle:= North_angle + 360;
2334 END
2335 Else NumberOfMLSequations:=0;
2336 END;
2337 IF ( Att.present = 1) And ( not att.flag) THEN
2338 BEGIN
2339     Rollangle:=Att.rollangle;
2340     Pitchangle:=Att.pitchangle;
2341 END;
2342 IF ( HDG.present = 1) And ( not hdg.flag) THEN Headingangle:=HDG.hdgangle;
2343 GPSPosition_Zerovector[1]:=pos_zerovector.x;
2344 GPSPosition_Zerovector[2]:=pos_zerovector.y;
2345 GPSPosition_Zerovector[3]:=pos_zerovector.z;
2346 Antenna_Zerovector[1]:=ant_zerovector.x;
2347 Antenna_Zerovector[2]:=ant_zerovector.y;
2348 Antenna_Zerovector[3]:=ant_zerovector.z;
2349
2350 present.att:= att.present;
2351 present.hdg:= hdg.present;
2352 present.gps:= gps.present;
2353 present.mls:= mls.present;
2354 present.dgps:= dgps.present;
2355 END;
2356 END;{PROCEDURE MRInterface}
2357
2358
2359
2360
2361
2362 Procedure CalcHybridPos( Var alldata: alldatatype; allowed_error: Double;
2363     Var position: positiontype);
2364
2365 {MAIN PROGRAMME}

```

```

2366 Var
2367 value      :      String;
2368
2369 BEGIN
2370 {-----}
2371 {Calculate the first eccentricity of the earth.}
2372
2373     e2:=(sqr(ae)-sqr(be))/sqr(ae);
2374
2375 {-----}
2376 {Set the measured values.}
2377
2378 Set_Measured_Values;
2379
2380 {-----}
2381 {The interface between the program of Marco Meijer and the program of René }
2382 {van Leeuwen.}
2383
2384 MRInterface(alldata);
2385
2386 position.flag:= True;
2387 If ( numberofgpsequations <> 0) Or ( numberofmlsequations <> 0)
2388 Then Begin
2389     gotoxy( 1,9);
2390     writeln( 'Number of GPS equations: ', numberofgpsequations,
2391         ' Number of MLS equations: ', numberofmlsequations);
2392     End;
2393
2394 If (( numberofgpsequations <4) And
2395     ( numberofmlsequations =0)) Or
2396     ( numberofgpsequations + numberofmlsequations <4) Or
2397     (( numberofgpsequations = 0) And
2398     ( numberofmlsequations <3))
2399 Then Exit;
2400
2401
2402
2403 {-----}
2404 Convergence_error:= allowed_error;
2405
2406 Datumpoint_ecef[ 1]:= alldata.mls.MLSthrespos.x;
2407 Datumpoint_ecef[ 2]:= alldata.mls.MLSthrespos.y;
2408 Datumpoint_ecef[ 3]:= alldata.mls.MLSthrespos.z;
2409
2410 Ecef_to_wgs84( datumpoint_ecef, datumpoint_wgs84);
2411
2412 EstWGS84.longitude:= position.wgs84lon * 180 / pi;
2413 If estwgs84.longitude > 0
2414 Then Begin
2415     estwgs84.hemilong:= 'east      ';
2416     estwgs84.longitude:= abs( estwgs84.longitude);
2417     End
2418 Else estwgs84.hemilong:= 'west      ';
2419
2420 EstWGS84.latitude := position.wgs84lat * 180 / pi;

```

```
2421 If estwgs84.latitude > 0
2422 Then Begin
2423     estwgs84.hemilat:= 'north    ';
2424     estwgs84.latitude:= abs( estwgs84.latitude);
2425 End
2426 Else estwgs84.hemilat:= 'south    ';
2427
2428 EstWGS84.height := position.wgs84alt;
2429
2430 Wgs84_to_Ecef( Estwgs84, Est_ecef);
2431
2432 Est_Local[1]:=1;
2433 Est_Local[2]:=1;
2434 Est_Local[3]:=1;
2435 Est_Local[4]:=0;
2436
2437 Calculate_Position(NumberOfGPSequations, {total of used GPS-equations}
2438                    gps1,gps2,gps3,gps4,gps5,gps6,gps7, {the used GPS-equations}
2439                    NumberOfMLSequations, {total of used MLS-equations}
2440                    mls1,mls2,mls3);      {the used MLS-equations    }
2441
2442 If iter >= maxiter
2443 Then position.flag:= True
2444 Else position.flag:= False;
2445
2446 If ( NumberOfMLSequations = 0) And ( NumberOfGPSequations >= 4)
2447 Then With position DO
2448     Begin
2449         x:=Est_ECEF[ 1];
2450         y:=Est_ECEF[ 2];
2451         z:=Est_ECEF[ 3];
2452         alldata.gps.DeltaT:= Est_Ecef[ 4];
2453         EcefTrueLocalFalse:= True;
2454         Convert_Pos_to_Wgs( position);
2455     End;
2456
2457 If (( NumberOfMLSequations + NumberOfGPSequations >= 4) And
2458     ( NumberOfMLSequations > 0)) Or
2459     ( numberofmlsequatons >= 3)
2460 Then With position Do
2461     Begin
2462         x:=Est_Local[ 1];
2463         y:=Est_Local[ 2];
2464         z:=Est_Local[ 3];
2465         alldata.gps.deltat:= Est_Local[ 4];
2466         EcefTrueLocalFalse:= False;
2467     End;
2468
2469 END;(MAIN PROGRAMME)
2470 End.
```

```

1 Unit GPScalc;
2 {*****}
3 { This unit provides procedures for the calculation of the SV position
4   for the GPS navigation system. It is only for L1 single frequency users}
5 {*****}
6
7 Interface
8
9 {$N+,E+}
10
11 Uses MIASglob, GPSglob;
12
13 Procedure Clockcorrection( sv: Byte; Var Gpsint: GPSinttype);
14 {*****}
15 { This procedure corrects the measured times for one GPS range which have an
16   error because of clock. The correction used is described in Appendix 3 to
17   Annex A to STANAG 4294, Draft issue L 1 August 1990.
18
19       Input : satellite number
20               ephemeris, clock and SV data
21               from Gpsint
22       Output: corrected GPStime }
23 {*****}
24
25
26 Procedure RelCorrection( sv: Byte; Var Gpsint: GPSinttype);
27 {*****}
28 { This procedure corrects the measured times for one GPS range which have an
29   error because of relativistic effects. The correction used is described in
30   Appendix 3 to Annex A to STANAG 4294, Draft issue L 1 August 1990.
31
32   To calculate Ek, a function from SVposition is used.
33       Input : satellite number
34               ephemeris, clock and SV data
35               from Gpsint
36       Output: corrected GPStime }
37 {*****}
38
39
40 Procedure L1correction( sv: Byte; Var Gpsint: GPSinttype);
41 {*****}
42 { This procedure corrects the measured times for on GPS range which have an
43   error because of group delay. The correction used is described in Appendix 3 to
44   Annex A to STANAG 4294, Draft issue L 1 August 1990.
45   Note: Do not use this for DGPS operation.
46       Input : satellite number
47               clockparameters for on SV
48       Output: corrected GPStime in Gpsint}
49 {*****}
50
51
52 Procedure Ionosphericcorrection( sv: Byte; position: positiontype;
53       Var Gpsint: GPSinttype);
54 {*****}
55 { This procedure corrects the GPStime for ionospheric delay. The formulas
56   used here are described in Appendix 6 to Annex A to STANAG 4294.
57   USE CLOCKCORRECTION ,L1CORRECTION AND ELEV_AZIM BEFORE USING THIS PROCEDURE,
58   BECAUSE THE GPSTIME, THE ELEVATION AND AZIMUTH MAY NOT BE CORRECT WHEN
59   USING THIS PROCEDURE FOR THE FIRST TIME.
60       Input : satellite number
61               position of the receiver
62               ionospheric data from Gpsint
63       Output: GPStime corrected in Gpsint}
64 {*****}
65
66
67 Procedure Troposphericcorrection( sv: Byte; pos: positiontype;
68       Var Gpsint: GPSinttype);
69 {*****}
70 { This procedure corrects the transmission time for tropospheric effects.
71   The range error because of troposphere is  $R(h, o) = f(o) * dR(h)$ .
72   The formulas used are from Appendix 6 to Annex A to STANAG 4294.
73   Note: use Elev_Azim before using Troposphericcorrection, because the ele-
74   vation is used here.
75       Input : satellite number
76               position of the receiver
77               elevation of user to sv
78       Output: corrected transmission time}
79 {*****}
80
81
82 Procedure SVposition( sv: Byte; Var Gpsint: GPSinttype);
83 {*****}
84 { This procedure calculates the position of a satellite. The formulas use
85   can be found in Appendix 3 to Annex A to STANAG 4294.
86       Input : satellitenumber
87               ephemerisdata
88               GPStime
89       Output: satellite position}
90 {*****}
91
92
93 Procedure SVpos_earthadjusted( sv: Byte; Var Gpsint: GPSinttype);
94 {*****}
95 { This procedure calculates the position of a satellite corrected for the
96   rotation of the earth during the signal transmission. The formulas used
97   can be found in ENAV II handout Winter 1991 from Ohio University.
98   The SV position is in an earth-centered-earth-fixed coordinate system.
99   This means that the satellite position at time of transmission is different
100  from the SVposition at time of reception (the earth rotated). To correct
101  for this, the earth rotation angle is calculated and the satellite
102  position is corrected for this rotation.
103       Input : satellitenumber
104               ephemerisdata
105               GPStime
106       Output: satellite position}
107 {*****}
108
109
110 Procedure Elev_Azim( sv: Byte; Var Gpsint: GPSinttype);

```

```

111 position: positiontype);
112 {*****}
113 { This procedure calculates the Azimuth and Elevation angle between
114 the satellite and the user. This procedure uses the satellite and
115 user position in ECEF coordinates and the userposition in WGS.
116 Note: Be sure that both WGS84 and Ecef fields are valid and correspond
117 to each other.
118 Input : satellitenumber
119         satelliteposition
120         userposition in ECEF
121         userposition in WGS
122 Output: azimuth and elevation.}
123 {*****}
124
125
126 Procedure Convert_Pos_to_WGS( Var position: positiontype);
127 {*****}
128 { Procedure for conversion of the receiver's ECEF-coordinates
129 (X,Y,Z) to <lat,long,alt> - coordinates.
130 dimension <m,m,m> ----> dimension <rad,rad,m>
131 This is an iterative procedure
132 Input : position, x,y,z
133 Output: position, lat, lon, h}
134 {*****}
135
136
137 Procedure Calc_PR( sv: Byte; Var Gpsint: GPSinttype);
138 {*****}
139 { This procedure calculates the Pseudo range from the measured transmission
140 and reception time.
141 Input : satellite number
142         Gpsint, tx and rx time
143 Output: Pseudo range}
144 {*****}
145
146
147 Procedure CalcSmoothPR( sv: Byte; Var Gpsint: GPSinttype);
148 {*****}
149 { This procedure calculates the smoothed Pseudo range from the previous
150 pseudoranges and integrated carrier phase. This procedure was written
151 by Peter Vianen. It is only meant for DGPS reference stations.
152 Input : satellite number
153         Gpsint, tx and rx time
154 Output: Pseudo range}
155 {*****}
156
157
158 Implementation
159
160 Uses Mathx;
161
162 Const
163 mu = 3.986005E14; {[m3/s2] WGS 84 value of earth's
164 universal gravitational constant}
165 c = 2.99792458E8; {[m/s] speed of light}
166
167 F = -4.442807633E-10; {[s/m1/2]}
168 omegaedot= 7.2921151467E-5; {[rad/s] WGS 84 value of the earth's
169 rotation rate}
170 flattening = 3.3528106E-3; {flattening of the earth }
171 earthAaxis = 6.378137E+6; {long earth axis }
172 earthEccentricitySqr = flattening * (2 - flattening); { e2 !! }
173 pi = 3.1415926535897932385;
174 tweepi = 2 * pi;
175 F_l1 = 1575420000; { L1 frequency}
176
177 Var
178 i : Integer;
179 k : Array [1..32] Of Longint;
180 previous_intcarphase,
181 DR_bias : Array [1..32] Of Double;
182
183
184 Procedure L1correction( sv: Byte; Var Gpsint: GPSinttype);
185
186 Begin
187 With Gpsint.prn[ sv] Do {tgps = tsv - (deltasv)L1}
188 txtime:= txtime + clock.Tgd;
189 {(deltasv)L1 = deltatsv - Tgd}
190 {so, tgps = tgps + Tgd}
191 End; { End of procedure L1correction}
192
193
194 Procedure Ionosphericcorrection( sv: Byte; position: positiontype;
195 Var Gpsint: GPSinttype);
196
197 Var
198 psi : Double; { Earth's central angle between user
199 position and Earth projection of
200 the ionospheric intersection point}
201 Geomaglat : Double; { Geomagnetic latitude of the earth}
202 { projection of the ionospheric }
203 { intersection point}
204 sqrgeomaglat : Double; { help variable, Sqr( Geomaglat)}
205 Geodetlat : Double; { Geodetic latitude of earth projec }
206 { tion of the ionospheric }
207 { intersection point}
208 Geodetlon : Double; { Geodetic longitude of earth projec }
209 { tion of the ionospheric }
210 { intersection point}
211 PER : Double; { period of model [s]}
212 t : Double; { Local time [s]}
213 X : Double; { Phase X [rad] }
214 SqrX : Double; { help variable, Sqr( X)}
215 AMP : Double; { vertical delay amplitude [s]}
216 F : Double; { Obliquity factor []}
217 Tiono : Double; { Ionospheric time correction [s]}
218
219 Begin
220 With Gpsint.prn[ sv] Do { calculate psi}

```

```

221     psi:= 0.0137 / ( E + 0.11) - 0.022;
222
223 With position Do
224 Begin
225     Geodetlat:= WGS84lat;
226     Geodetlon:= WGS84lon;
227 End;
228
229 With Gpsint.prn[ sv] Do
230 Begin
231     Geodetlat:= Geodetlat + psi * Cos ( A);
232     Geodetlon:= Geodetlon + psi * sin ( A) / cos ( Geodetlat);
233 End;
234
235 Geomaglat:= Geodetlat + 0.064 * ( Geodetlon - 1.617);
236
237     { PER = sum from 0 to including 3 of }
238 With Gpsint.iono Do      { betan* Geomaglat^n}
239 Begin
240     sqrgeomaglat:= Sqr( Geomaglat);
241     PER:= beta0 + beta1 * Geomaglat + beta2 * sqrgeomaglat +
242         beta3 * sqrgeomaglat * Geomaglat;
243 End;
244 If PER < 72000
245 Then PER:= 72000;
246
247 With Gpsint.prn[ sv] Do      { calculate Local time}
248     t:= 4.32E4 * Geodetlon + txtime;
249 If t >= 86400
250 Then t:= t - 86400;
251 If t < 0
252 Then t:= t + 86400;
253
254 X:= 2 * pi * ( t - 50400) / PER;  { calculate phase}
255
256     { AMP = sum from 0 to including 3 of}
257 With Gpsint. iono Do      { alfan * Geomaglat^n}
258     AMP := alfa0 + alfa1 * Geomaglat + alfa2 * sqrgeomaglat +
259         alfa3 * sqrgeomaglat * Geomaglat;
260 If AMP < 0
261 Then AMP := 0;
262
263 With Gpsint. prn[ sv] Do
264     F := 1 + 16 * Sqr( 0.53 - E) * ( 0.53 - E);
265
266     sqrx:= Sqr( X);
267 If Abs( X) < 1.57
268 Then Tiono:= F * ( 5E-9 + AMP * (1 - SqrX/2 + Sqr( SqrX)/24))
269 Else Tiono:= F * 5E-9;
270
271 With Gpsint.prn[ sv] Do      { (deltatsv)I = deltatsv - Tiono}
272     txtime:= txtime + Tiono;      { so tGPS = tGPS + Tiono}
273 End;      { End of procedure Ionosphericcorrection}
274
275

```

```

276 Procedure Troposphericcorrection( sv: Byte; pos: positiontype;
277     Var Gpsint: GPSinttype);
278
279 Const
280     Ns      =      324.8;      { Surface refractivity index at MSL}
281                                     { is 324.8 with standard dev of 25.98}
282                                     { If measured, it can be used}
283 Var
284     f      :      Double;      { Range error as function of
285                                     elevation angle}
286     dN      :      Double;
287     dR      :      Double;      { Range error as a function of
288                                     altitude [m]}
289     temp     :      Double;      { temporary variable}
290     N1      :      Double;      { refractivity index at h=1km}
291
292 Begin
293     dN      :=      -7.32 * exp( 0.005577 * Ns);
294
295 With pos Do
296 Begin
297     If (h >=0) And ( h < 1)
298     Then dR:= (
299         Ns * ( 1 - h) +
300         0.5 * dN * ( 1 - Sqr( h)) + 1430 + 732
301     ) * 1E-3;
302
303     If (h > 1) And ( h <= 9)
304     Then Begin
305         N1:= Ns + dN;
306         temp:= ln( N1 / 105)/8;
307         dR:= (
308             (- N1/ temp) *
309             (
310                 exp( ( 1 - 9) * temp) - exp( ( 1 - h) * temp)
311             ) + 732
312         ) * 1E-3;
313     End;
314
315     If (h > 9)
316     Then dR:= (
317         (-105/0.1424) *
318         (
319             exp( -0.1424 * ( 20186.8 - 9)) -
320             exp( -0.1424 * ( h - 9))
321         )
322     ) * 1E-3;
323 End;
324
325 With Gpsint. prn[ sv] Do
326 Begin
327     If E < 90
328     Then f:= 1/
329         (
330             sin( E) + 0.00143/

```

```

331      (
332      tan( E) + 0.0455
333      )
334      )
335      Else f:= 1;
336
337      txttime:= txttime + ( f * dR / c); { correct GPStime}
338      { Range error/speed is time [s]}
339      End;
340 End; {End of procedure Troposphericcorrection}
341
342
343 Function EccentricAnomaly( Mk: Double; Var Gpsint: GPSinttype;
344      sv: byte): Double;
345 {*****}
346 { This function solves the eccentricAnomaly ( Ek) by iteration. The
347 equation to be solved is : Ek = Mk + e sin( Ek). So the iteration formula
348 becomes: Ekn+1 = Mk + e * sin( Ekn) which we iterate, until the difference
349 between Ekn+1 and Ekn is smaller than epsilon. We start with Ekn = Mk.
350      Input : Mean Anomaly Mk
351      Eccentricity from Gpsint
352      Output: Eccentric Anomaly Ek }
353 {*****}
354 Const
355      epsilon      =      1E-9; { accuracy for Newton Raphson}
356      Maxiter      =      20; { maximum number of iterations}
357 Var
358      Ekn          :      Double; { for iteration}
359      Eknplus1     :      Double;
360      iter         :      Integer; { number of iterations}
361
362 Begin
363      With Gpsint.prn[ sv] Do
364      With ephemeris Do
365      Begin
366      Eknplus1:= Mk; { starting condition}
367      iter:= 0;
368      Repeat
369      Ekn:= Eknplus1;
370      Eknplus1 := Mk + e * sin( Ekn); { iterative equation}
371      Inc( iter);
372      Until ( Abs( Eknplus1 - Ekn) < epsilon) And (iter <= Maxiter);
373      { stop criterion}
374      If (iter > Maxiter)
375      Then Gpsint.prn[ sv].flag:= True; { not enough convergence}
376      End;
377      EccentricAnomaly:= Eknplus1;
378 End; { End function eccentricanomaly}
379
380
381 Procedure SVposition( sv: Byte; Var Gpsint: GPSinttype);
382
383 Var
384      Ax          :      Double; {Semi-Major Axis}
385      no          :      Double; {computed mean motion}

```

```

386      tk          :      Double; {time from Ephemeris Reference Epoch}
387      t           :      Double; {GPStime at time of transmission
388      GPStime corrected for transit time
389      (rangs/speed of light)}
390      n           :      Double; {corrected mean motion}
391      Mk          :      Double; {Mean Anomaly}
392      vk          :      Double; {True Anomaly}
393      vknum,      :      Double; {numerator for calculating Vk}
394      vkden       :      Double; {denominator for calculating Vk}
395      phik,       :      Double; {argument of latitude}
396      twophik,    :      Double; {2 * phik}
397      sintwophik, :      Double; { sin of twophik}
398      costwophik, :      Double; { cos of twophik}
399      deltau,     :      Double; {argument of latitude correction}
400      deltark,    :      Double; {radius correction}
401      deltaik,    :      Double; {correction to inclination}
402      uk,         :      Double; {corrected argument of latitude}
403      rk,         :      Double; {corrected radius}
404      ik,         :      Double; {corrected inclination}
405      cosik,      :      Double; {cos( ik)}
406      sinik,      :      Double; {sin( ik)}
407      xk1,        :      Double; {positions in orbital plane}
408      yk1,        :      Double;
409      cosomegak,  :      Double; {cos( omegak)}
410      sinomegak,  :      Double; {sin( omegak)}
411      omegak      :      Double; {corrected longitude of
412      ascending node}
413
414
415 Begin { begin procedure SVposition}
416      With Gpsint.prn[ sv] Do { calculate t }
417      Begin
418      t := txttime;
419      With ephemeris Do
420      Begin
421      tk := t - toe;
422      If ( tk > 302400)
423      Then tk:= tk - 604800;
424      If ( tk < -302400)
425      Then tk:= tk + 604800;
426
427      Ax := Sqr( Asqrt);
428      If (Ax = 0)
429      Then Exit;
430
431      no := Sqrt( mu/ Ax) / Ax;
432
433      n := no + deltan * pi;
434
435      Mk := Mo * pi + n * tk;
436
437      Ek := EccentricAnomaly( Mk, Gpsint, sv);
438      { Compute the true anomaly}
439      vknum := Sqrt
440      (

```

```

441      1 - Sqr( e)
442    ) *
443    sin( Ek);           {calculate vk in parts}
444
445    vkden := cos( Ek) - e;
446
447    If Abs( vkden) < 1E-100      { protect against division by zero}
448    Then If vknum >= 0
449      Then vk:= pi * 0.5
450      Else vk:= pi * 1.5
451    Else vk:= arctan( vknum / vkden);
452
453    If vknum < 0                { 3rd or 4th quadrant}
454    Then If vkden < 0
455      Then vk:= vk + pi        { 3rd quadrant}
456      Else vk:= vk + 2 * pi    { 4th quadrant}
457    Else If vkden < 0          { 1st or 2nd quadrant}
458      Then vk:= vk + pi;       { 2nd quadrant}
459
460
461    phik := vk + omega * pi;
462
463                                { Compute the 2nd harmonic
464                                perturbations}
465
466    twophik:= 2 * phik;
467    sintwophik:= sin( twophik);
468    costwophik:= cos( twophik);
469    deltauk:= Cus * sintwophik + Cuc * costwophik;
470    deltark:= Crc * costwophik + Crs * sintwophik;
471    deltaik:= Cic * costwophik + Cis * sintwophik;{corrected}
472
473    uk := phik + deltauk;
474    rk := Ax *
475      (
476        1 - e * cos( Ek)
477      ) + deltark;
478    ik := (io + IDOT * tk) * pi + deltaik ;
479
480    xk1 := rk * cos( uk);
481    yk1 := rk * sin( uk);
482
483                                { correct longitude of
484                                the ascending node}
485    omegak := omegao * pi + ( omegadot * pi - omegaedot) * tk -
486      omegaedot * toe;
487
488  End;
489
490  With position Do
491    Begin
492      sinomegak:= sin( omegak);
493      cosomegak:= cos( omegak);
494      cosik := cos( ik);
495      sinik := sin( ik);
496      x := xk1 * cosomegak - yk1 * cosik * sinomegak;
497      y := xk1 * sinomegak + yk1 * cosik * cosomegak;
498      z := yk1 * sinik;
499    End;

```

[illegible]

```

551 End;          { End of procedure clockcorrection}
552
553
554 Procedure RelCorrection( sv: Byte; Var Gpsint: GPSinttype);
555
556 Var
557   t,
558   tk,
559   deltatr      : Double;
560   Ax,
561   no,
562   n,
563   Mk           : Double;
564
565 Begin
566   With Gpsint.prn[ sv].ephemeris Do
567     deltatr:= F * e * Asqrt;
568
569   With Gpsint.prn[ sv] Do          { calculate Ek}
570     Begin
571       t := txttime;
572       With ephemeris Do
573         Begin
574           tk := t - toe;
575           If ( tk > 302400)
576             Then tk:= tk - 604800;
577           If ( tk < -302400)
578             Then tk:= tk + 604800;
579
580           Ax := Sqr( Asqrt);
581           If (Ax = 0)
582             Then Exit;
583
584           no := Sqr( mu/ Ax) / Ax;
585
586           n := no + deltan * pi;
587
588           Mk := Mo * pi + n * tk;
589         End;
590       End;
591
592   Gpsint.prn[sv].Ek:= EccentricAnomaly( Mk, Gpsint, sv);
593
594   With Gpsint.prn[ sv] Do
595     deltatr:= deltatr * sin ( Ek);
596
597   With Gpsint.prn[ sv] Do
598     txttime:= txttime - deltatr; {delete received txttime, insert
599                               GPStime at time of transmission}
600 End;
601
602
603 Procedure Elev_Azim( sv: Byte; Var Gpsint: GPSinttype;
604                     position: positiontype);
605 Var
606   xvec,          { vector coordinates from user}
607   yvec,          { to satellite}
608   zvec           : Double;
609   xrot,
610   yrot           : Double; { transformed( rotated) coordinates}
611   vec           : Double; { vector length}
612   temp          : Double; { help variable}
613   sintheta,
614   costheta,
615   sinpsi,
616   cospsi        : Double; { helpvariables}
617
618 Begin
619   With Gpsint. prn[ sv].position Do
620     Begin
621       xvec := x - position.x;
622       yvec := y - position.y;
623       zvec := z - position.z;
624       costheta:= cos( position.WGS84lon);
625       sintheta:= sin( position.WGS84lon);
626       cospsi := cos( position.WGS84lat);
627       sinpsi := sin( position.WGS84lat);
628     End;
629
630   vec := Sqr( Sqr( xvec) + Sqr( yvec) + Sqr( zvec));
631   temp := xvec * costheta + yvec * sintheta;
632   xrot := temp * sinpsi - zvec * cospsi;
633   yrot := yvec * cospsi - xvec * sinpsi;
634
635   With Gpsint. prn[ sv] Do
636     Begin
637       E:= arcsin( (zvec * sinpsi + temp * cospsi)/ vec);
638       A:= arctan( yrot / -xrot);
639
640       If ( xrot < 0) And ( yrot < 0)
641         Then A:= A + tweepi;
642
643       If ( xrot < 0) And ( yrot >= 0)
644         Then A:= A;
645
646       If ( xrot >= 0) And ( yrot < 0)
647         Then A:= pi - A;
648
649       If ( xrot >= 0) And ( yrot >= 0)
650         Then A:= A + pi;
651     End;
652 End;
653
654
655 procedure Convert_Pos_to_WGS( Var position: positiontype);
656
657 const
658   smallValue = 1E-10; { small value to check if user is at a pole, }
659                     { requested accuracy of NewtonRaphson etc. }
660   verySmall  = 1E-15; { small value to prevent denominator = 0 }

```



```

661 maxIterAllowed = 25;    { maximum number of iterations allowed }
662
663 var
664   numberOfRuns           : byte;
665   C,func1,Delta_lat      : Double;
666
667 begin
668   With position Do
669     Begin
670       { secures against crash on Poles }
671       If (Abs( X ) < smallValue) And
672         (Abs( Y ) < smallValue)
673       Then Begin           { user is at or close to North/South Pole }
674         If ( Z > 0 ) Then
675           WGS84lat := pi/2           { North Pole }
676         Else
677           WGS84lat := -pi/2;        { South Pole }
678
679         { EXPERIMENTAL }
680         { the value of the longitude is calculated }
681         { as zero, but with help of the course }
682         { this value must be defined yet !!!!!!!!! }
683         WGS84lon := 0;
684         WGS84alt := Abs( Z ) - (EarthAxis *
685                               sqrt(1 - earthEccentricitySqr) );
686
687       End;
688
689       { calculating the longitude }
690       If ( Abs( X ) < verySmall ) then
691       Begin           { protects arctan function against zero X }
692         If X >= 0 then X := verySmall;
693         If X < 0 then X := -verySmall;
694       End;
695       WGS84lon := arctan( Y / X );
696       { correction of the arctan function: changes }
697       { range from -pi/2 -> pi/2 to -pi -> pi. }
698       If ( ( X < 0 ) And ( Y > 0 ) )
699       Then WGS84lon := WGS84lon + pi;
700
701       If ( ( X < 0 ) And ( Y < 0 ) )
702       Then WGS84lon := WGS84lon - pi;
703
704       { calculation of the latitude with Newton-Raphson }
705       C := Sqrt( Sqr( X ) + Sqr( Y ) );
706       WGS84lat := arctan( Z / C );
707       numberOfRuns := 1;
708       Delta_lat := 10;           { initial value }
709
710       While ( Abs( Delta_lat ) > smallValue ) And
711         ( numberOfRuns < maxIterAllowed ) Do
712       Begin
713         If WGS84lat = 0.5*pi
714         Then WGS84lat := 0.5*pi - smallValue;

```

```

716
717       func1 := 1 / Sqrt( 1 - earthEccentricitySqr *
718         Sqr( sin( WGS84lat ) ) );
719
720       Delta_lat := ( Z - C * sin( WGS84lat ) /
721         cos( WGS84lat ) + earthEccentricitySqr *
722         EarthAxis * func1 * sin( WGS84lat ) ) /
723         ( C / Sqr( cos( WGS84lat ) ) - EarthAxis *
724         cos( WGS84lat ) * earthEccentricitySqr *
725         func1 * ( 1 + earthEccentricitySqr *
726         Sqr( func1 * sin( WGS84lat ) ) ) );
727
728       WGS84lat := WGS84lat + Delta_lat;
729       numberOfRuns := numberOfRuns + 1;
730     End;
731
732     If numberOfRuns >= maxIterAllowed
733     Then flag := True;           { failure in Newton Raphson }
734
735     { calculation of the altitude }
736     WGS84alt := X / ( cos( WGS84lat ) *
737       cos( WGS84lon ) ) - earthAxis /
738       Sqrt( 1 - earthEccentricitySqr *
739       Sqr( sin( WGS84lat ) ) );
740   End; { end with position }
741 End; { of procedure Convert_Pos_To_WGS }
742
743 Procedure Calc_PR( sv: Byte; Var Gpsint: GPSinttype);
744
745 Begin
746   With Gpsint.prn[ sv ] Do
747     pr:= (rxtime - txtime) * c;
748   End;
749
750 Procedure CalcSmoothPR ( sv: Byte; Var Gpsint: GPSinttype);
751
752 Var
753   PR_measured,
754   delta_phase,
755   DR
756   : Double;
757
758 Begin
759   With Gpsint.prn[sv] Do
760     Begin
761       PR_measured := ( rxtime - ( txtime + ( intcarphase / F_l1 ) ) ) * c;
762
763       If ( k[sv] = 0 )
764       Then PR := PR_measured
765       Else Begin
766         delta_phase := intcarphase - previous_intcarphase [sv];
767         DR := delta_phase * c / F_l1;
768         PR := ( ( PR + DR ) * k[sv] + PR_measured ) / ( k[sv] + 1 );
769         DR_bias[sv] := ( PR - PR_measured + k[sv] * DR_bias[sv] )

```

```
771                / ( k[sv] + 1 );
772            PR := PR - DR_bias[sv];
773        End;
774
775        previous_intcarphase[sv] := intcarphase;
776        k[sv] := k[sv] + 1;
777    End;
778 End;
779
780
781
782 Begin
783     For i := 1 To 32 Do
784         Begin
785             DR_bias [i] := 0;
786             k [i] := 0;
787         End;
788     End;
789 End.
```

```

1 Unit Mathx;
2
3 Interface
4
5 {$N+,E+}
6
7 Uses MIAsglob;
8
9 Const
10  pi      =      3.1415926535897932385;
11  tweepi  =      2 * pi;
12
13
14 Function tan( arg: Double): Double;
15 {*****}
16 { This function provides the tangens of the argument. It is not very accurate
17   Input : argument in radians
18   Output: tangens of the argument}
19 {*****}
20
21
22 Function arccos( x: Double): Double;
23 {*****}
24 { This function provides the arccosine of the argument
25   Input : argument in radians
26   Output: arccosine of the argument}
27 {*****}
28
29 Function arcsin( x: Double): Double;
30 {*****}
31 { This function provides the arcsine of the argument
32   Input : argument in radians
33   Output: arcsine of the argument}
34 {*****}
35
36
37 Implementation
38
39
40 Function tan( arg: Double): Double;
41
42 Var
43  si :      Double;      { temporary sine}
44  co :      Double;      { temporary cosine}
45
46 Begin
47  si:= sin( arg);      { tan x = sin x/ cos x}
48  co:= cos( arg);
49
50  If (co = 0)
51  Then If si >= 0
52    Then tan:= 1E38
53    Else tan:= -1E38
54  Else tan := si / co;
55 End;
56
57
58
59 Function arccos( x: Double): Double;
60
61 begin
62  if (X = 0) then
63    arccos := pi/2
64  else
65    begin
66
67    if X < 0 then
68      begin
69        if sqr(X) > 1 then
70          X := -1.0;
71        arccos := arctan ( ( sqrt( 1 - sqr(X) ) ) / X ) + pi;
72      end
73    else
74      begin
75        if sqr(X) > 1 then
76          X := 1;
77        arccos := arctan ( ( sqrt( 1 - sqr(X) ) ) / X );
78      end;
79    end;
80 end; (* of function arccos *)
81
82
83 Function arcsin( x: Double): Double;
84
85 begin
86  If x > 1
87  Then x:= 1;
88
89  If x < -1
90  Then x:= -1;
91
92  if ( abs( X ) = 1 ) then
93    begin
94      if X = 1 then
95        arcsin := pi/2
96      else
97        arcsin := -pi/2;
98    end
99  else
100    arcsin := arctan ( X / ( sqrt( 1 - sqr(X) ) ));
101 end; (* of function arcsin *)
102
103
104 End.
```

```

1 unit Matrix;
2
3 {-----}
4 {-}
5 {- Turbo Pascal Numerical Methods Toolbox}
6 {- Copyright (c) 1986, 87 by Borland International, Inc.}
7 {-}
8 {- This unit provides procedures for dealing with systems of linear}
9 {- equations.}
10 {-}
11 {-----}
12
13 {$N+,E+}
14
15 interface
16
17 Uses MIAsglob;
18
19 {$IFOPT N+}
20 type
21   Float = Double; { 8 byte real, requires 8087 math chip }
22
23 const
24   TNNearlyZero = 1E-015;
25 {$ELSE}
26 type
27   Float = real; { 6 byte real, no math chip required }
28
29 const
30   TNNearlyZero = 1E-07;
31 {$ENDIF}
32
33   TNArrSize = 10; { Size of the matrix }
34
35 type
36   TNvector = array[1..TNArrSize] of Float;
37   TNmatrix = array[1..TNArrSize] of TNvector;
38
39 procedure Determinant(Dimen : integer;
40   Data : TNmatrix;
41   var Det : Float;
42   var Error : byte);
43
44 {-----}
45 {-}
46 {- Input: Dimen, Data}
47 {- Output: Det, Error}
48 {-}
49 {- Purpose : Calculate the determinant of a matrix by}
50 {- making it upper-triangular and then}
51 {- taking the product of the diagonal elements.}
52 {-}
53 {- User-defined Types : TNvector = array[1..TNArrSize] of real;
54 {- TNmatrix = array[1..TNArrSize] of TNvector}
55 {-}
56 {- Global Variables : Dimen : integer; Dimension of the square matrix}
57 {- Data : TNmatrix; Square matrix}
58 {- Det : real; Determinant of Data}
59 {- Error : integer; Flags if something goes wrong}
60 {-}
61 {- Errors : 0: No errors;
62 {- 1: Dimen < 1}
63 {-}
64 {-----}
65
66 procedure Inverse(Dimen : integer;
67   Data : TNmatrix;
68   var Inv : TNmatrix;
69   var Error : byte);
70
71 {-----}
72 {-}
73 {- Input: Dimen, Data}
74 {- Output: Inv, Error}
75 {-}
76 {- Purpose : calculate the inverse of a matrix with}
77 {- Gauss-Jordan elimination.}
78 {-}
79 {- User-defined Types : TNvector = array[1..TNArrSize] of real;
80 {- TNmatrix = array[1..TNArrSize] of TNvector}
81 {-}
82 {- Global Variables : Dimen : integer; Dimension of the square matrix}
83 {- Data : TNmatrix; Square matrix}
84 {- Inv : TNmatrix; Inverse of Data}
85 {- Error : integer; Flags if something goes wrong}
86 {-}
87 {- Errors : 0: No errors;
88 {- 1: Dimen < 1}
89 {- 2: no inverse exists}
90 {-}
91 {-----}
92
93 procedure Gaussian_Elimination(Dimen : integer;
94   Coefficients : TNmatrix;
95   Constants : TNvector;
96   var Solution : TNvector;
97   var Error : byte);
98
99 {-----}
100 {-}
101 {- Input: Dimen, Coefficients, Constants}
102 {- Output: Solution, Error}
103 {-}
104 {- Purpose : Calculate the solution of a linear set of}
105 {- equations using Gaussian elimination and}
106 {- backwards substitution.}
107 {-}
108 {- User-defined Types : TNvector = array[1..TNArrSize] of real
109 {- TNmatrix = array[1..TNArrSize] of TNvector}
110 {-}

```

```

111 {- Global Variables : Dimen : integer;      Dimension of the square  ->
112 {-                               matrix      ->
113 {-                               Coefficients : TNmatrix; Square matrix ->
114 {-                               Constants : TNvector; Constants of each equation->
115 {-                               Solution : TNvector; Unique solution to the  ->
116 {-                               set of equations ->
117 {-                               Error : integer; Flags if something goes  ->
118 {-                               wrong.      ->
119 {-                               ->
120 {-                               Errors: 0: No errors;      ->
121 {-                               1: Dimen < 1      ->
122 {-                               2: no solution exists      ->
123 {-                               ->
124 {-                               ->
125 {------}
126
127 procedure Partial_Pivoting(Dimen      : integer;
128                               Coefficients : TNmatrix;
129                               Constants : TNvector;
130                               var Solution : TNvector;
131                               var Error : byte);
132
133 {------}
134 {-                               ->
135 {-                               Input: Dimen, Coefficients, Constants ->
136 {-                               Output: Solution, Error      ->
137 {-                               ->
138 {-                               Purpose : Calculate the solution of a linear set of ->
139 {-                               equations using Gaussian elimination, maximal ->
140 {-                               pivoting and backwards substitution. ->
141 {-                               ->
142 {-                               User-defined Types : TNvector = array[1..TNArraySize] of real; ->
143 {-                               TNmatrix = array[1..TNArraySize] of TNvector ->
144 {-                               ->
145 {-                               Global Variables : Dimen      : integer; Dimen of the square ->
146 {-                               matrix      ->
147 {-                               Coefficients : TNmatrix; Square matrix ->
148 {-                               Constants : TNvector; Constants of each equation->
149 {-                               Solution : TNvector; Unique solution to the  ->
150 {-                               set of equations ->
151 {-                               Error : integer; Flags if something goes  ->
152 {-                               wrong.      ->
153 {-                               ->
154 {-                               Errors : 0: No errors;      ->
155 {-                               1: Dimen < 2      ->
156 {-                               2: no solution exists      ->
157 {-                               ->
158 {------}
159
160 procedure LU_Decompose(Dimen      : integer;
161                               Coefficients : TNmatrix;
162                               var Decomp : TNmatrix;
163                               var Permute : TNmatrix;
164                               var Error : byte);
165
166 {------}
167 {-                               ->
168 {-                               Input: Dimen, Coefficients      ->
169 {-                               Output: Decomp, Permute, Error ->
170 {-                               ->
171 {-                               Purpose : Decompose a square matrix into an upper ->
172 {-                               triangular and lower triangular matrix such that ->
173 {-                               the product of the two triangular matrices is ->
174 {-                               the original matrix. This procedure also returns ->
175 {-                               a permutation matrix which records the ->
176 {-                               permutations resulting from partial pivoting. ->
177 {-                               ->
178 {-                               User-defined Types : TNvector = array[1..TNArraySize] of real ->
179 {-                               TNmatrix = array[1..TNArraySize] of TNvector ->
180 {-                               ->
181 {-                               Global Variables : Dimen      : integer; Dimen of the coefficients ->
182 {-                               Matrix      ->
183 {-                               Coefficients : TNmatrix; Coefficients matrix ->
184 {-                               Decomp : TNmatrix; Decomposition of ->
185 {-                               Coefficients matrix ->
186 {-                               Permute : TNmatrix; Record of partial ->
187 {-                               Pivoting ->
188 {-                               Error : integer; Flags if something goes  ->
189 {-                               wrong.      ->
190 {-                               ->
191 {-                               Errors : 0: No errors;      ->
192 {-                               1: Dimen < 1      ->
193 {-                               2: No decomposition possible; singular matrix ->
194 {-                               ->
195 {------}
196
197 procedure LU_Solve(Dimen      : integer;
198                               var Decomp : TNmatrix;
199                               Constants : TNvector;
200                               var Permute : TNmatrix;
201                               var Solution : TNvector;
202                               var Error : byte);
203 {------}
204 {-                               ->
205 {-                               Input: Dimen, Decomp, Constants, Permute ->
206 {-                               Output: Solution, Error      ->
207 {-                               ->
208 {-                               Purpose : Calculate the solution of a linear set of ->
209 {-                               equations using an LU decomposed matrix, a ->
210 {-                               permutation matrix and backwards and forward ->
211 {-                               substitution. ->
212 {-                               ->
213 {-                               User_defined Types : TNvector = array[1..TNArraySize] of real ->
214 {-                               TNmatrix = array[1..TNArraySize] of TNvector ->
215 {-                               ->
216 {-                               Global Variables : Dimen      : integer; Dimen of the square ->
217 {-                               matrix      ->
218 {-                               Decomp : TNmatrix; Decomposition of ->
219 {-                               coefficient matrix ->
220 {-                               Constants : TNvector; Constants of each equation ->

```

```

221 {-          Permute   : TNmatrix; Permutation matrix from  ->
222 {-          partial pivoting  ->
223 {-          Solution   : TNvector; Unique solution to the  ->
224 {-          set of equations  ->
225 {-          Error      : integer;  Flags if something goes  ->
226 {-          wrong.        ->
227 {-          ->
228 {-          Errors : 0: No errors;  ->
229 {-          1: Dimen < 1  ->
230 {-          ->
231 {------}
232
233 procedure Gauss_Seidel(Dimen      : integer;
234          Coefficients : TNmatrix;
235          Constants    : TNvector;
236          Tol          : Float;
237          MaxIter       : integer;
238          var Solution  : TNvector;
239          var Iter      : integer;
240          var Error     : byte);
241
242 {------}
243 {-          ->
244 {-          Input: Dimen, Coefficients, Constants, Tol, MaxIter  ->
245 {-          Output: Solution, Iter, Error  ->
246 {-          ->
247 {-          Purpose : Calculate the solution of a linear set of  ->
248 {-          equations using Gauss - Seidel iteration.  ->
249 {-          ->
250 {-          User-defined Types : TNvector = array[1..TNArraySize] of real  ->
251 {-          TNmatrix = array[1..TNArraySize] of TNvector  ->
252 {-          ->
253 {-          Global Variables : Dimen : integer;      Dimen of the square  ->
254 {-          matrix  ->
255 {-          Coefficients : TNmatrix; Square matrix  ->
256 {-          Constants   : TNvector; Constants of each equation-}
257 {-          Tol         : real; Tolerance in answer  ->
258 {-          MaxIter     : integer; Maximum number of  ->
259 {-          iterations allowed  ->
260 {-          Solution    : TNvector; Unique solution to the  ->
261 {-          set of equations  ->
262 {-          Iter        : integer; Number of iterations  ->
263 {-          Error       : integer; Flags if something goes  ->
264 {-          wrong.      ->
265 {-          ->
266 {-          Errors : 0: No errors;  ->
267 {-          1: Iter >= MaxIter and matrix not  ->
268 {-          diagonally dominant  ->
269 {-          2: Iter >= MaxIter  ->
270 {-          3: Dimen < 1  ->
271 {-          4: Tol <= 0  ->
272 {-          5: MaxIter < 0  ->
273 {-          6: Zero on the diagonal of  ->
274 {-          the Coefficients matrix  ->
275 {-          7: Diverging  ->

```

```

276 {-          ->
277 {-          Note: If the Gauss-Seidel iterative method is  ->
278 {-          applied to an underdetermined system of equations  ->
279 {-          (i.e. one of the equations is a linear  ->
280 {-          superposition of the others) it will converge  ->
281 {-          to a (non-unique) solution. The Gauss-Seidel  ->
282 {-          method is unable to distinguish between unique  ->
283 {-          and non-unique solutions.  ->
284 {-          If you are concerned that your equations  ->
285 {-          may be underdetermined, solve them with  ->
286 {-          Gaussian elimination (GAUSELIM.INC or  ->
287 {-          PARTPIVT.INC  ->
288 {-          ->
289 {------}
290
291 implementation
292
293 {$I Matrix.inc}
294
295
296 end. { Matrix }

```

```

1 {-----}
2 {-                                     -}
3 {-   Turbo Pascal Numerical Methods Toolbox   -}
4 {-   Copyright (c) 1986, 87 by Borland International, Inc.   -}
5 {-                                     -}
6 {-----}
7
8 procedure Determinant(Dimen : integer;
9                      Data : TNmatrix;
10                     var Det : Float;
11                     var Error : byte);
12
13 procedure Initial(Dimen : integer;
14                  var Data : TNmatrix;
15                  var Det : Float;
16                  var Error : byte);
17
18 {-----}
19 {- This procedure tests for errors in the value of Dimen -}
20 {-----}
21
22 begin
23   Error := 0;
24   if Dimen < 1 then
25     Error := 1
26   else
27     if Dimen = 1 then
28       Det := Data[1, 1];
29     end; { procedure Initial }
30
31 procedure EROswitch(var Row1 : TNvector;
32                    var Row2 : TNvector);
33
34 {-----}
35 {- Elementary row operation - switching two rows -}
36 {-----}
37
38 var
39   DummyRow : TNvector;
40
41 begin
42   DummyRow := Row1;
43   Row1 := Row2;
44   Row2 := DummyRow;
45 end; { procedure EROswitch }
46
47 procedure EROmultAdd(Multiplier : Float;
48                     Dimen : integer;
49                     var ReferenceRow : TNvector;
50                     var ChangingRow : TNvector);
51
52 {-----}
53 {- Row operation - adding a multiple of one row to another -}
54 {-----}
55
56 var
57   Term : integer;
58
59 begin
60   for Term := 1 to Dimen do
61     ChangingRow[Term] := ChangingRow[Term] + Multiplier * ReferenceRow[Term];
62   end; { procedure EROmultAdd }
63
64 function Deter(Dimen : integer;
65               var Data : TNmatrix) : Float;
66
67 {-----}
68 {- Input: Dimen, Data                                     -}
69 {- Output: Deter                                          -}
70 {- Function returns the determinant of the Data matrix -}
71 {-----}
72
73
74 var
75   PartialDeter, Multiplier : Float;
76   Row, ReferenceRow : integer;
77   DetEqualsZero : boolean;
78
79 procedure Pivot(Dimen : integer;
80                 ReferenceRow : integer;
81                 var Data : TNmatrix;
82                 var PartialDeter : Float;
83                 var DetEqualsZero : boolean);
84
85 {-----}
86 {- Input: Dimen, ReferenceRow, Data, PartialDeter        -}
87 {- Output: Data, PartialDeter, DetEqualsZero             -}
88 {- This procedure searches the ReferenceRow column of the -}
89 {- matrix Data for the first non-zero element below the  -}
90 {- diagonal. If it finds one, then the procedure switches -}
91 {- rows so that the non-zero element is on the diagonal. -}
92 {- Switching rows changes the determinant by a factor of  -}
93 {- -1; this change is returned in PartialDeter.          -}
94 {- If it doesn't find one, the matrix is singular and the -}
95 {- Determinant is zero (DetEqualsZero = true is returned). -}
96 {-----}
97
98
99 var
100   NewRow : integer;
101
102 begin
103   DetEqualsZero := true;
104   NewRow := ReferenceRow;
105   while DetEqualsZero and (NewRow < Dimen) do { Try to find a row }
106     { with a non-zero }
107     { element in this }
108     { column          }
109   begin
110     NewRow := Succ(NewRow);

```

```

111   if ABS(Data[NewRow, ReferenceRow]) > TNNearlyZero then
112   begin
113       EROswitch(Data[NewRow], Data[ReferenceRow]);
114       { Switch these two rows }
115       DetEqualsZero := false;
116       PartialDeter := -PartialDeter; { Switching rows changes }
117                                   { the determinant by a }
118                                   { factor of -1 }
119   end;
120 end;
121 end; { procedure Pivot }
122
123 begin { function Deter }
124   DetEqualsZero := false;
125   PartialDeter := 1;
126   ReferenceRow := 0;
127   { Make the matrix upper triangular }
128   while not(DetEqualsZero) and (ReferenceRow < Dimen - 1) do
129   begin
130       ReferenceRow := Succ(ReferenceRow);
131       { If diagonal element is zero then switch rows }
132       if ABS(Data[ReferenceRow, ReferenceRow]) < TNNearlyZero then
133         Pivot(Dimen, ReferenceRow, Data, PartialDeter, DetEqualsZero);
134       if not(DetEqualsZero) then
135         for Row := ReferenceRow + 1 to Dimen do
136           { Make the ReferenceRow element of this row zero }
137           if ABS(Data[Row, ReferenceRow]) > TNNearlyZero then
138             begin
139               Multiplier := -Data[Row, ReferenceRow] /
140                           Data[ReferenceRow, ReferenceRow];
141               EROmultAdd(Multiplier, Dimen, Data[ReferenceRow], Data[Row]);
142             end;
143           { Multiply the diagonal Term into PartialDeter }
144           PartialDeter := PartialDeter * Data[ReferenceRow, ReferenceRow];
145         end;
146       if DetEqualsZero then
147         Deter := 0
148       else
149         Deter := PartialDeter * Data[Dimen, Dimen];
150     end; { function Deter }
151
152   begin { procedure Determinant }
153     Initial(Dimen, Data, Det, Error);
154     if Dimen > 1 then
155       Det := Deter(Dimen, Data);
156     end; { procedure Determinant }
157
158   procedure Inverse((Dimen : integer;
159                     Data : TNmatrix;
160                     var Inv : TNmatrix;
161                     var Error : byte));
162
163
164   procedure Initial(Dimen : integer;
165                     var Data : TNmatrix;

```

```

166                     var Inv : TNmatrix;
167                     var Error : byte);
168
169   {-----}
170   {- Input: Dimen, Data -}
171   {- Output: Inv, Error -}
172   {- This procedure test for errors in the value of Dimen -}
173   {-----}
174
175   var
176     Row : integer;
177
178   begin
179     Error := 0;
180     if Dimen < 1 then
181       Error := 1
182     else
183       begin
184         { First make the inverse-to-be the identity matrix }
185         FillChar(Inv, SizeOf(Inv), 0);
186         for Row := 1 to Dimen do
187           Inv[Row, Row] := 1;
188         if Dimen = 1 then
189           if ABS(Data[1, 1]) < TNNearlyZero then
190             Error := 2 { Singular matrix }
191           else
192             Inv[1, 1] := 1 / Data[1, 1];
193         end;
194       end;
195     end; { procedure Initial }
196
197   procedure EROdiv(Divisor : Float;
198                   Dimen : integer;
199                   var Row : TNvector);
200
201   {-----}
202   {- Input: Divisor, Dimen, Row -}
203   {- elementary row operation - dividing by a constant -}
204   {-----}
205
206   var
207     Term : integer;
208
209   begin
210     for Term := 1 to Dimen do
211       Row[Term] := Row[Term] / Divisor;
212     end; { procedure EROdiv }
213
214   procedure EROswitch(var Row1 : TNvector;
215                       var Row2 : TNvector);
216
217   {-----}
218   {- Input: Row1, Row2 -}
219   {- Output: Row1, Row2 -}

```



```

221 {-                                     -}
222 {- Elementary row operation - switching two rows -}
223 {------}
224
225 var
226   DummyRow : TNvector;
227
228 begin
229   DummyRow := Row1;
230   Row1 := Row2;
231   Row2 := DummyRow;
232 end; { procedure EROswitch }
233
234 procedure EROmultAdd(Multiplier : Float;
235                     Dimen      : integer;
236                     var ReferenceRow : TNvector;
237                     var ChangingRow : TNvector);
238
239 {------}
240 {- Input: Multiplier, Dimen, ReferenceRow, ChangingRow -}
241 {- Output: ChangingRow -}
242 {- -}
243 {- Row operation - adding a multiple of one row to another -}
244 {------}
245
246 var
247   Term : integer;
248
249 begin
250   for Term := 1 to Dimen do
251     ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
252   end; { procedure EROmultAdd }
253
254
255 procedure Inver(Dimen : integer;
256               var Data : TNmatrix;
257               var Inv : TNmatrix;
258               var Error : byte);
259
260 {------}
261 {- Input: Dimen, Data -}
262 {- Output: Inv, Error -}
263 {- -}
264 {- This procedure computes the inverse of the matrix Data -}
265 {- and stores it in the matrix Inv. If the matrix Data -}
266 {- is singular, then Error = 2 is returned. -}
267 {------}
268
269 var
270   Divisor, Multiplier : Float;
271   Row, ReferenceRow : integer;
272
273 procedure Pivot(Dimen : integer;
274               ReferenceRow : integer;
275               var Data : TNmatrix;
```

```

276               var Inv : TNmatrix;
277               var Error : byte);
278
279 {------}
280 {- Input: Dimen, ReferenceRow, Data, Inv -}
281 {- Output: Data, Inv, Error -}
282 {- -}
283 {- This procedure searches the ReferenceRow column of -}
284 {- the Data matrix for the first non-zero element below -}
285 {- the diagonal. If it finds one, then the procedure -}
286 {- switches rows so that the non-zero element is on the -}
287 {- diagonal. This same operation is applied to the Inv -}
288 {- matrix. If no non-zero element exists in a column, the -}
289 {- matrix is singular and no inverse exists. -}
290 {------}
291
292 var
293   NewRow : integer;
294
295 begin
296   Error := 2; { No inverse exists }
297   NewRow := ReferenceRow;
298   while (Error > 0) and (NewRow < Dimen) do
299     { Try to find a }
300     { row with a non-zero }
301     { diagonal element }
302     begin
303       NewRow := Succ(NewRow);
304       if ABS(Data[NewRow, ReferenceRow]) > TNNearlyZero then
305         begin
306           EROswitch(Data[NewRow], Data[ReferenceRow]);
307           { Switch these two rows }
308           EROswitch(Inv[NewRow], Inv[ReferenceRow]);
309           Error := 0;
310         end;
311     end; { while }
312 end; { procedure Pivot }
313
314 begin { procedure Inver }
315   { Make Data matrix upper triangular }
316   ReferenceRow := 0;
317   while (Error = 0) and (ReferenceRow < Dimen) do
318     begin
319       ReferenceRow := Succ(ReferenceRow);
320       { Check to see if the diagonal element is zero }
321       if ABS(Data[ReferenceRow, ReferenceRow]) < TNNearlyZero then
322         Pivot(Dimen, ReferenceRow, Data, Inv, Error);
323       if Error = 0 then
324         begin
325           Divisor := Data[ReferenceRow, ReferenceRow];
326           EROdiv(Divisor, Dimen, Data[ReferenceRow]);
327           EROdiv(Divisor, Dimen, Inv[ReferenceRow]);
328           for Row := 1 to Dimen do
329             { Make the ReferenceRow element of this row zero }
330             if (Row <> ReferenceRow) and
```

```

331      (ABS(Data[Row, ReferenceRow]) > TNNearlyZero) then
332      begin
333          Multiplier := -Data[Row, ReferenceRow] /
334              Data[ReferenceRow, ReferenceRow];
335          EROmultAdd(Multiplier, Dimen, Data[ReferenceRow], Data[Row]);
336          EROmultAdd(Multiplier, Dimen, Inv[ReferenceRow], Inv[Row]);
337      end;
338  end;
339  end;
340 end; { procedure Inver }
341
342 begin { procedure Inverse }
343   Initial(Dimen, Data, Inv, Error);
344   if Dimen > 1 then
345       Inver(Dimen, Data, Inv, Error);
346 end; { procedure Inverse }
347
348 procedure Gaussian_Elimination(Dimen      : integer;
349                                Coefficients : TNmatrix;
350                                Constants    : TNvector;
351                                var Solution  : TNvector;
352                                var Error     : byte);
353
354 procedure Initial(Dimen      : integer;
355                  var Coefficients : TNmatrix;
356                  var Constants   : TNvector;
357                  var Solution    : TNvector;
358                  var Error       : byte);
359
360 {-----}
361 {- Input: Dimen, Coefficients, Constants      -}
362 {- Output: Solution, Error                    -}
363 {-
364 {- This procedure test for errors in the value of Dimen. -}
365 {- This procedure also finds the solution for the      -}
366 {- trivial case Dimen = 1.                            -}
367 {-----}
368
369 begin
370   Error := 0;
371   if Dimen < 1 then
372       Error := 1
373   else
374       if Dimen = 1 then
375           if ABS(Coefficients[1, 1]) < TNNearlyZero then
376               Error := 2
377           else
378               Solution[1] := Constants[1] / Coefficients[1, 1];
379 end; { procedure Initial }
380
381 procedure EROswitch(var Row1 : TNvector;
382                   var Row2 : TNvector);
383
384 {-----}
385 {- Input: Row1, Row2                      -}

```

```

386 {- Output: Row1, Row2                      -}
387 {-
388 {- elementary row operation - switching two rows -}
389 {-----}
390
391 var
392   DummyRow : TNvector;
393
394 begin
395   DummyRow := Row1;
396   Row1 := Row2;
397   Row2 := DummyRow;
398 end; { procedure EROswitch }
399
400 procedure EROmultAdd(Multiplier : Float;
401                     Dimen       : integer;
402                     var ReferenceRow : TNvector;
403                     var ChangingRow : TNvector);
404
405 {-----}
406 {- Input: Multiplier, Dimen, ReferenceRow, ChangingRow -}
407 {- Output: ChangingRow                                -}
408 {-
409 {- row operation - adding a multiple of one row to another -}
410 {-----}
411
412 var
413   Term : integer;
414
415 begin
416   for Term := 1 to Dimen do
417       ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
418   end; { procedure EROmultAdd }
419
420 procedure UpperTriangular(Dimen      : integer;
421                          var Coefficients : TNmatrix;
422                          var Constants   : TNvector;
423                          var Error       : byte);
424
425 {-----}
426 {- Input: Dimen, Coefficients, Constants      -}
427 {- Output: Coefficients, Constants, Error     -}
428 {-
429 {- This procedure makes the coefficient matrix upper triangular. -}
430 {- The operations which perform this are also performed on the -}
431 {- Constants vector.                                           -}
432 {- If one of the main diagonal elements of the upper triangular -}
433 {- matrix is zero, then the Coefficients matrix is singular and -}
434 {- no solution exists (Error = 2 is returned).                -}
435 {-----}
436
437 var
438   Multiplier : Float;
439   Row, ReferenceRow : integer;
440

```

```

441 procedure Pivot(Dimen      : integer;
442                 ReferenceRow : integer;
443                 var Coefficients : TNmatrix;
444                 var Constants : TNvector;
445                 var Error      : byte);
446
447 {-----}
448 {- Input: Dimen, ReferenceRow, Coefficients -}
449 {- Output: Coefficients, Constants, Error -}
450 {- -}
451 {- This procedure searches the ReferenceRow column of the -}
452 {- Coefficients matrix for the first non-zero element below -}
453 {- the diagonal. If it finds one, then the procedure switches -}
454 {- rows so that the non-zero element is on the diagonal. -}
455 {- It also switches the corresponding elements in the -}
456 {- Constants vector. If it doesn't find one, the matrix is -}
457 {- singular and no solution exists (Error = 2 is returned). -}
458 {-----}
459
460 var
461   NewRow : integer;
462   Dummy : Float;
463
464 begin
465   Error := 2;      { No solution exists }
466   NewRow := ReferenceRow;
467   while (Error > 0) and (NewRow < Dimen) do { Try to find a
468                                           { row with a non-zero }
469                                           { diagonal element }
470   begin
471     NewRow := Succ(NewRow);
472     if ABS(Coefficients[NewRow, ReferenceRow]) > TNNearlyZero then
473     begin
474       EROswitch(Coefficients[NewRow], Coefficients[ReferenceRow]);
475       { Switch these two rows }
476       Dummy := Constants[NewRow];
477       Constants[NewRow] := Constants[ReferenceRow];
478       Constants[ReferenceRow] := Dummy;
479       Error := 0; { Solution may exist }
480     end;
481   end;
482 end; { procedure Pivot }
483
484 begin { procedure UpperTriangular }
485   ReferenceRow := 0;
486   while (Error = 0) and (ReferenceRow < Dimen - 1) do
487   begin
488     ReferenceRow := Succ(ReferenceRow);
489     { Check to see if the main diagonal element is zero }
490     if ABS(Coefficients[ReferenceRow, ReferenceRow]) < TNNearlyZero then
491       Pivot(Dimen, ReferenceRow, Coefficients, Constants, Error);
492     if Error = 0 then
493       for Row := ReferenceRow + 1 to Dimen do
494         { Make the ReferenceRow element of this row zero }
495         if ABS(Coefficients[Row, ReferenceRow]) > TNNearlyZero then

```

```

496   begin
497     Multiplier := -Coefficients[Row, ReferenceRow] /
498                 Coefficients[ReferenceRow, ReferenceRow];
499     EROmultAdd(Multiplier, Dimen,
500               Coefficients[ReferenceRow], Coefficients[Row]);
501     Constants[Row] := Constants[Row] +
502                     Multiplier * Constants[ReferenceRow];
503   end;
504 end; { while }
505 if ABS(Coefficients[Dimen, Dimen]) < TNNearlyZero then
506   Error := 2; { No solution }
507 end; { procedure UpperTriangular }
508
509 procedure BackwardsSub(Dimen      : integer;
510                       var Coefficients : TNmatrix;
511                       var Constants : TNvector;
512                       var Solution : TNvector);
513
514 {-----}
515 {- Input: Dimen, Coefficients, Constants -}
516 {- Output: Solution -}
517 {- -}
518 {- This procedure applies backwards substitution to the upper -}
519 {- triangular Coefficients matrix and Constants vector. The -}
520 {- resulting vector is the solution to the set of equations and -}
521 {- is returned in the vector Solution. -}
522 {-----}
523
524 var
525   Term, Row : integer;
526   Sum : Float;
527
528 begin
529   Term := Dimen;
530   while Term >= 1 do
531   begin
532     Sum := 0;
533     for Row := Term + 1 to Dimen do
534       Sum := Sum + Coefficients[Term, Row] * Solution[Row];
535     Solution[Term] := (Constants[Term] - Sum) / Coefficients[Term, Term];
536     Term := Pred(Term);
537   end;
538 end; { procedure BackwardsSub }
539
540 begin { procedure Gaussian_Elimination }
541   Initial(Dimen, Coefficients, Constants, Solution, Error);
542   if Dimen > 1 then
543   begin
544     UpperTriangular(Dimen, Coefficients, Constants, Error);
545     if Error = 0 then
546       BackwardsSub(Dimen, Coefficients, Constants, Solution);
547   end;
548 end; { procedure Gaussian_Elimination }
549
550 procedure Partial_Pivoting((Dimen      : integer;

```

```

551         Coefficients : TNmatrix;
552         Constants    : TNvector;
553         var Solution  : TNvector;
554         var Error     : byte);
555
556
557 procedure Initial(Dimen      : integer;
558                 var Coefficients : TNmatrix;
559                 var Constants   : TNvector;
560                 var Solution    : TNvector;
561                 var Error      : byte);
562
563 {-----}
564 {- Input: Dimen, Coefficients, Constants -}
565 {- Output: Solution, Error -}
566 {-}
567 {- This procedure test for errors in the value of Dimen. -}
568 {- This procedure also finds the solution for the -}
569 {- trivial case Dimen = 1. -}
570 {-----}
571
572 begin
573     Error := 0;
574     if Dimen < 1 then
575         Error := 1
576     else
577         if Dimen = 1 then
578             if ABS(Coefficients[1, 1]) < TNNearlyZero then
579                 Error := 2
580             else
581                 Solution[1] := Constants[1] / Coefficients[1, 1];
582 end; { procedure Initial }
583
584 procedure EROswitch(var Row1 : TNvector;
585                   var Row2 : TNvector);
586
587 {-----}
588 {- Input: Row1, Row2 -}
589 {- Output: Row1, Row2 -}
590 {-}
591 {- elementary row operation - switching two rows -}
592 {-----}
593
594 var
595     DummyRow : TNvector;
596
597 begin
598     DummyRow := Row1;
599     Row1 := Row2;
600     Row2 := DummyRow;
601 end; { procedure EROswitch }
602
603 procedure EROmultAdd(Multiplier : Float;
604                   Dimen      : integer;
605                   var ReferenceRow : TNvector;
```

```

606                   var ChangingRow : TNvector);
607
608 {-----}
609 {- Input: Multiplier, Dimen, ReferenceRow, ChangingRow -}
610 {- Output: ChangingRow -}
611 {-}
612 {- Row operation - adding a multiple of one row to another -}
613 {-----}
614
615 var
616     Term : integer;
617
618 begin
619     for Term := 1 to Dimen do
620         ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
621 end; { procedure EROmultAdd }
622
623 procedure UpperTriangular(Dimen      : integer;
624                         var Coefficients : TNmatrix;
625                         var Constants   : TNvector;
626                         var Error      : byte);
627
628 {-----}
629 {- Input: Dimen, Coefficients, Constants -}
630 {- Output: Coefficients, Constants, Error -}
631 {-}
632 {- This procedure makes the coefficient matrix upper triangular. -}
633 {- The operations which perform this are also performed on the -}
634 {- Constants vector. -}
635 {- If one of the main diagonal elements of the upper triangular -}
636 {- matrix is zero, then the Coefficients matrix is singular and -}
637 {- no solution exists (Error = 2 is returned). -}
638 {-----}
639
640 var
641     Multiplier : Float;
642     Row, ReferenceRow : integer;
643
644 procedure Pivot(Dimen      : integer;
645               ReferenceRow : integer;
646               var Coefficients : TNmatrix;
647               var Constants   : TNvector;
648               var Error      : byte);
649
650 {-----}
651 {- Input: Dimen, ReferenceRow, Coefficients -}
652 {- Output: Coefficients, Constants, Error -}
653 {-}
654 {- This procedure searches the ReferenceRow column of the -}
655 {- Coefficients matrix for the largest non-zero element below -}
656 {- the diagonal. If it finds one, then the procedure switches -}
657 {- rows so that the largest non-zero element is on the -}
658 {- diagonal. It also switches the corresponding elements in -}
659 {- the Constants vector. If it doesn't find a non-zero element, -}
660 {- the matrix is singular and no solution exists -}

```

```

661 (- (Error = 2 is returned).          -)
662 (------)
663
664 var
665   PivotRow, Row : integer;
666   Dummy : Float;
667
668 begin
669   { First, find the row with the largest element }
670   PivotRow := ReferenceRow;
671   for Row := ReferenceRow + 1 to Dimen do
672     if ABS(Coefficients[Row, ReferenceRow]) >
673       ABS(Coefficients[PivotRow, ReferenceRow]) then
674       PivotRow := Row;
675   if PivotRow <> ReferenceRow then
676     { Second, switch these two rows }
677     begin
678       EROswitch(Coefficients[PivotRow], Coefficients[ReferenceRow]);
679       Dummy := Constants[PivotRow];
680       Constants[PivotRow] := Constants[ReferenceRow];
681       Constants[ReferenceRow] := Dummy;
682     end
683   else { If the diagonal element is zero, no solution exists }
684     if ABS(Coefficients[ReferenceRow, ReferenceRow]) < TNNearlyZero then
685       Error := 2; { No solution }
686   end; { procedure Pivot }
687
688 begin { procedure UpperTriangular }
689   { Make Coefficients matrix upper triangular }
690   ReferenceRow := 0;
691   while (Error = 0) and (ReferenceRow < Dimen - 1) do
692     begin
693       ReferenceRow := Succ(ReferenceRow);
694       { Find row with largest element in this column }
695       { and switch this row with the ReferenceRow }
696       Pivot(Dimen, ReferenceRow, Coefficients, Constants, Error);
697       if Error = 0 then
698         for Row := ReferenceRow + 1 to Dimen do
699           { Make the ReferenceRow element of these rows zero }
700           if ABS(Coefficients[Row, ReferenceRow]) > TNNearlyZero then
701             begin
702               Multiplier := -Coefficients[Row, ReferenceRow] /
703                 Coefficients[ReferenceRow, ReferenceRow];
704               EROmultAdd(Multiplier, Dimen,
705                 Coefficients[ReferenceRow], Coefficients[Row]);
706               Constants[Row] := Constants[Row] +
707                 Multiplier * Constants[ReferenceRow];
708             end;
709         end;
710       if ABS(Coefficients[Dimen, Dimen]) < TNNearlyZero then
711         Error := 2; { No solution }
712     end; { procedure UpperTriangular }
713
714 procedure BackwardsSub(Dimen : integer;
715   var Coefficients : TNmatrix;

```

```

716   var Constants : TNvector;
717   var Solution : TNvector);
718
719 (------)
720 {- Input: Dimen, Coefficients, Constants -}
721 {- Output: Solution -}
722 {- -}
723 {- This procedure applies backwards substitution to the upper -}
724 {- triangular Coefficients matrix and Constants vector. The -}
725 {- resulting vector is the solution to the set of equations and -}
726 {- is stored in the vector Solution. -}
727 (------)
728
729 var
730   Term, Row : integer;
731   Sum : Float;
732
733 begin
734   Term := Dimen;
735   while Term >= 1 do
736     begin
737       Sum := 0;
738       for Row := Term + 1 to Dimen do
739         Sum := Sum + Coefficients[Term, Row] * Solution[Row];
740       Solution[Term] := (Constants[Term] - Sum) / Coefficients[Term, Term];
741       Term := Pred(Term);
742     end;
743   end; { procedure BackwardsSub }
744
745 begin { procedure Partial_Pivoting }
746   Initial(Dimen, Coefficients, Constants, Solution, Error);
747   if Dimen > 1 then
748     begin
749       UpperTriangular(Dimen, Coefficients, Constants, Error);
750       if Error = 0 then
751         BackwardsSub(Dimen, Coefficients, Constants, Solution);
752     end;
753   end; { procedure Partial_Pivoting }
754
755 procedure LU_Decompose((Dimen : integer;
756   Coefficients : TNmatrix;
757   var Decomp : TNmatrix;
758   var Permute : TNmatrix;
759   var Error : byte));
760
761
762 procedure TestInput(Dimen : integer;
763   var Error : byte);
764
765 (------)
766 {- Input: Dimen -}
767 {- Output: Error -}
768 {- -}
769 {- This procedure checks to see if the -}
770 {- value of Dimen is greater than 1. -}

```

```

771 {-----}
772
773 begin
774   Error := 0;
775   if Dimen < 1 then
776     Error := 1;
777 end; { procedure TestInput }
778
779 function RowColumnMult(Row   : integer;
780                        var Lower : TNmatrix;
781                        Column : integer;
782                        var Upper : TNmatrix) : Float;
783
784 {-----}
785 {- Input: Row, Lower, Column, Upper -}
786 {- Function return: dot product of row Row of Lower -}
787 {-                  and column Column of Upper -}
788 {-----}
789
790 var
791   Term : integer;
792   Sum : Float;
793
794 begin
795   Sum := 0;
796   for Term := 1 to Row - 1 do
797     Sum := Sum + Lower[Row, Term] * Upper[Term, Column];
798   RowColumnMult := Sum;
799 end; { function RowColumnMult }
800
801
802 procedure Pivot(Dimen      : integer;
803                ReferenceRow : integer;
804                var Coefficients : TNmatrix;
805                var Lower       : TNmatrix;
806                var Upper       : TNmatrix;
807                var Permuted    : TNmatrix;
808                var Error       : byte);
809
810 {-----}
811 {- Input: Dimen, ReferenceRow, Coefficients, -}
812 {-        Lower, Upper, Permuted -}
813 {- Output: Coefficients, Lower, Permuted, Error -}
814 {- -}
815 {- This procedure searches the ReferenceRow column of the -}
816 {- Coefficients matrix for the element in the Row below the -}
817 {- main diagonal which produces the largest value of -}
818 {- -}
819 {-      Coefficients[Row, ReferenceRow] -}
820 {- -}
821 {-      Sum K=1 to ReferenceRow - 1 of -}
822 {-      Upper[Row, k] - Lower[k, ReferenceRow] -}
823 {- -}
824 {- If it finds one, then the procedure switches -}
825 {- rows so that this element is on the main diagonal. The -}

```

```

826 {- procedure also switches the corresponding elements in the -}
827 {- Permute matrix and the Lower matrix. If the largest value of -}
828 {- the above expression is zero, then the matrix is singular -}
829 {- and no solution exists (Error = 2 is returned). -}
830 {-----}
831
832 var
833   PivotRow, Row : integer;
834   ColumnMax, TestMax : Float;
835
836 procedure EROswitch(var Row1 : TNvector;
837                    var Row2 : TNvector);
838
839 {-----}
840 {- Input: Row1, Row2 -}
841 {- Output: Row1, Row2 -}
842 {- -}
843 {- elementary row operation - switching two rows -}
844 {-----}
845
846 var
847   DummyRow : TNvector;
848
849 begin
850   DummyRow := Row1;
851   Row1 := Row2;
852   Row2 := DummyRow;
853 end; { procedure EROswitch }
854
855 begin { procedure Pivot }
856   { First, find the row with the largest TestMax }
857   PivotRow := ReferenceRow;
858   ColumnMax := ABS(Coefficients[ReferenceRow, ReferenceRow] -
859                   RowColumnMult(ReferenceRow, Lower, ReferenceRow, Upper));
860   for Row := ReferenceRow + 1 to Dimen do
861     begin
862       TestMax := ABS(Coefficients[Row, ReferenceRow] -
863                     RowColumnMult(Row, Lower, ReferenceRow, Upper));
864
865       if TestMax > ColumnMax then
866         begin
867           PivotRow := Row;
868           ColumnMax := TestMax;
869         end;
870     end;
871
872   if PivotRow <> ReferenceRow then
873     { Second, switch these two rows }
874     begin
875       EROswitch(Coefficients[PivotRow], Coefficients[ReferenceRow]);
876       EROswitch(Lower[PivotRow], Lower[ReferenceRow]);
877       EROswitch(Permuted[PivotRow], Permuted[ReferenceRow]);
878     end
879   else
880     { If ColumnMax is zero, no solution exists }

```

```

881   if ColumnMax < TNNearlyZero then
882     Error := 2;    { No solution exists }
883 end; { procedure Pivot }
884
885 procedure Decompose(Dimen      : integer;
886                    var Coefficients : TNmatrix;
887                    var Decomp      : TNmatrix;
888                    var Permute     : TNmatrix;
889                    var Error       : byte);
890
891 {-----}
892 {- Input: Dimen, Coefficients          -}
893 {- Output: Decomp, Permute, Error      -}
894 {-                                     -}
895 {- This procedure decomposes the Coefficients matrix -}
896 {- into two triangular matrices, a lower and an upper -}
897 {- one. The lower and upper matrices are combined -}
898 {- into one matrix, Decomp. The permutation matrix, -}
899 {- Permute, records the effects of partial pivoting. -}
900 {-----}
901
902 var
903   Upper, Lower : TNmatrix;
904   Term, Index : integer;
905
906 procedure Initialize(Dimen : integer;
907                    var Lower : TNmatrix;
908                    var Upper : TNmatrix;
909                    var Permute : TNmatrix);
910
911 {-----}
912 {- Output: Dimen, Lower, Upper, Permute -}
913 {-                                     -}
914 {- This procedure initializes the above variables. -}
915 {- Lower and Upper are initialized to the zero -}
916 {- matrix and Diag is initialized to the identity -}
917 {- matrix. -}
918 {-----}
919
920 var
921   Diag : integer;
922
923 begin
924   FillChar(Upper, SizeOf(Upper), 0);
925   FillChar(Lower, SizeOf(Lower), 0);
926   FillChar(Permute, SizeOf(Permute), 0);
927   for Diag := 1 to Dimen do
928     Permute[Diag, Diag] := 1;
929 end; { procedure Initialize }
930
931 begin { procedure Decompose }
932   Initialize(Dimen, Lower, Upper, Permute);
933
934   { partial pivoting on row 1 }
935   Pivot(Dimen, 1, Coefficients, Lower, Upper, Permute, Error);

```

```

936   if Error = 0 then
937   begin
938     Lower[1, 1] := 1;
939     Upper[1, 1] := Coefficients[1, 1];
940
941     for Term := 1 to Dimen do
942     begin
943       Lower[Term, 1] := Coefficients[Term, 1] / Upper[1, 1];
944       Upper[1, Term] := Coefficients[1, Term] / Lower[1, 1];
945     end;
946   end;
947
948   Term := 1;
949   while (Error = 0) and (Term < Dimen - 1) do
950   begin
951     Term := Succ(Term);
952
953     { perform partial pivoting on row Term }
954     Pivot(Dimen, Term, Coefficients, Lower, Upper, Permute, Error);
955
956     Lower[Term, Term] := 1;
957     Upper[Term, Term] := Coefficients[Term, Term] -
958                        RowColumnMult(Term, Lower, Term, Upper);
959
960     if ABS(Upper[Term, Term]) < TNNearlyZero then
961       Error := 2 { no solutions }
962     else
963       for Index := Term + 1 to Dimen do
964       begin
965         Upper[Term, Index] := Coefficients[Term, Index] -
966                             RowColumnMult(Term, Lower, Index, Upper);
967         Lower[Index, Term] := (Coefficients[Index, Term] -
968                             RowColumnMult(Index, Lower, Term, Upper)) /
969                             Upper[Term, Term];
970       end;
971     end;
972
973     Lower[Dimen, Dimen] := 1;
974     Upper[Dimen, Dimen] := Coefficients[Dimen, Dimen] -
975                        RowColumnMult(Dimen, Lower, Dimen, Upper);
976     if ABS(Upper[Dimen, Dimen]) < TNNearlyZero then
977       Error := 2;
978     { Combine the upper and lower triangular matrices into one }
979     Decomp := Upper;
980
981     for Term := 2 to Dimen do
982       for Index := 1 to Term - 1 do
983         Decomp[Term, Index] := Lower[Term, Index];
984   end; { procedure Decompose }
985
986 begin { procedure LU_Decompose }
987   TestInput(Dimen, Error);
988   if Error = 0 then
989     if Dimen = 1 then
990     begin

```

```

991     Decomp := Coefficients;
992     Permute[1, 1] := 1;
993   end
994   else
995     Decompose(Dimen, Coefficients, Decomp, Permute, Error);
996 end; { procedure LU_Decompose }
997
998 procedure LU_Solve((Dimen      : integer;
999                   var Decomp   : TNmatrix;
1000                   Constants : TNvector;
1001                   var Permute  : TNmatrix;
1002                   var Solution : TNvector;
1003                   var Error    : byte));
1004
1005 procedure Initial(Dimen      : integer;
1006                 var Solution : TNvector;
1007                 var Error    : byte);
1008
1009 {-----}
1010 {- Input: Dimen -}
1011 {- Output: Solution, Error -}
1012 {- -}
1013 {- This procedure initializes the Solution vector. -}
1014 {- It also checks to see if the value of Dimen is -}
1015 {- greater than 1. -}
1016 {-----}
1017
1018 begin
1019   Error := 0;
1020   FillChar(Solution, SizeOf(Solution), 0);
1021   if Dimen < 1 then
1022     Error := 1;
1023 end; { procedure Initial }
1024
1025 procedure FindSolution(Dimen      : integer;
1026                      var Decomp   : TNmatrix;
1027                      var Constants : TNvector;
1028                      var Solution : TNvector);
1029
1030 {-----}
1031 {- Input: Dimen, Decomp, Constants -}
1032 {- Output: Solution -}
1033 {- -}
1034 {- The Decomp matrix contains a lower and upper triangular -}
1035 {- matrix. -}
1036 {- This procedure performs a two step backwards substitution -}
1037 {- to compute the solution to the system of equations. First, -}
1038 {- forward substitution is applied to the lower triangular -}
1039 {- matrix and Constants vector yielding PartialSolution. Then -}
1040 {- backwards substitution is applied to the Upper matrix and -}
1041 {- the PartialSolution vector yielding Solution. -}
1042 {-----}
1043
1044 var
1045   PartialSolution : TNvector;
1046   Term, Index : integer;
1047   Sum : Float;
1048
1049 begin { procedure FindSolution }
1050   { First solve the lower triangular matrix }
1051   PartialSolution[1] := Constants[1];
1052   for Term := 2 to Dimen do
1053     begin
1054       Sum := 0;
1055       for Index := 1 to Term - 1 do
1056         if Term = Index then
1057           Sum := Sum + PartialSolution[Index];
1058         else
1059           Sum := Sum + Decomp[Term, Index] * PartialSolution[Index];
1060       PartialSolution[Term] := Constants[Term] - Sum;
1061     end;
1062   { Then solve the upper triangular matrix }
1063   Solution[Dimen] := PartialSolution[Dimen] / Decomp[Dimen, Dimen];
1064   for Term := Dimen - 1 downto 1 do
1065     begin
1066       Sum := 0;
1067       for Index := Term + 1 to Dimen do
1068         Sum := Sum + Decomp[Term, Index] * Solution[Index];
1069       Solution[Term] := (PartialSolution[Term] - Sum)/Decomp[Term, Term];
1070     end;
1071 end; { procedure FindSolution }
1072
1073 procedure PermuteConstants(Dimen      : integer;
1074                          var Permute  : TNmatrix;
1075                          var Constants : TNvector);
1076
1077 var
1078   Row, Column : integer;
1079   Entry : Float;
1080   TempConstants : TNvector;
1081
1082 begin
1083   for Row := 1 to Dimen do
1084     begin
1085       Entry := 0;
1086       for Column := 1 to Dimen do
1087         Entry := Entry + Permute[Row, Column] * Constants[Column];
1088       TempConstants[Row] := Entry;
1089     end;
1090   Constants := TempConstants;
1091 end; { procedure PermuteConstants }
1092
1093 begin { procedure Solve_LU_Decompostion }
1094   Initial(Dimen, Solution, Error);
1095   if Error = 0 then
1096     PermuteConstants(Dimen, Permute, Constants);
1097     FindSolution(Dimen, Decomp, Constants, Solution);
1098 end; { procedure LU_Solve }
1099
1100 procedure Gauss_Seidel((Dimen      : integer;

```



```

1101      Coefficients : TNmatrix;
1102      Constants   : TNvector;
1103      Tol          : Float;
1104      MaxIter      : integer;
1105      var Solution  : TNvector;
1106      var Iter      : integer;
1107      var Error     : byte));
1108
1109
1110 var
1111   Guess : TNvector;
1112
1113 procedure TestInput(Dimen      : integer;
1114                    Tol         : Float;
1115                    MaxIter     : integer;
1116                    var Coefficients : TNmatrix;
1117                    var Constants  : TNvector;
1118                    var Solution   : TNvector;
1119                    var Error      : byte);
1120
1121 {-----}
1122 {- Input: Dimen, Tol, MaxIter -}
1123 {- Coefficients, -}
1124 {- Constants -}
1125 {- Output: Solution, Error -}
1126 {-}
1127 {- test the input data for errors -}
1128 {- The procedure also finds the -}
1129 {- solution for the trivial case -}
1130 {- Dimen = 0. -}
1131 {-----}
1132
1133 begin
1134   Error := 0;
1135   if Dimen < 1 then
1136     Error := 3
1137   else
1138     if Tol <= 0 then
1139       Error := 4
1140     else
1141       if MaxIter < 0 then
1142         Error := 5;
1143       if (Error = 0) and (Dimen = 1) then
1144         begin
1145           if ABS(Coefficients[1, 1]) < TNNearlyZero then
1146             Error := 6
1147           else
1148             Solution[1] := Constants[1] / Coefficients[1, 1];
1149         end;
1150       end; { procedure TestInput }
1151
1152 procedure TestForDiagDominance(Dimen      : integer;
1153                               var Coefficients : TNmatrix;
1154                               var Error      : byte);
1155
1156 {-----}
1157 {- Input: Dimen, Coefficients -}
1158 {- Output: Error -}
1159 {-}
1160 {- This procedure examines the Coefficients matrix to see if it is -}
1161 {- diagonally dominant. If it is, then the Gauss-Seidel iterative -}
1162 {- method will converge to a solution of this system of equations; -}
1163 {- if not, then convergence may not be possible with this method -}
1164 {- and Error = 1 (which is a warning) is returned. If one of the -}
1165 {- elements on the main diagonal of the Coefficients matrix is -}
1166 {- zero, then the matrix is singular and cannot be solved and -}
1167 {- Error = 6 is returned. In such a case, one of the direct -}
1168 {- methods for solving systems of equations (e.g. Gaussian -}
1169 {- elimination) should be used. -}
1170 {-----}
1171
1172 var
1173   Row, Column : integer;
1174   Sum : Float;
1175
1176 begin
1177   Row := 0;
1178   while (Row < Dimen) and (Error < 2) do
1179     begin
1180       Row := Succ(Row);
1181       Sum := 0;
1182       for Column := 1 to Dimen do
1183         if Column <> Row then
1184           Sum := Sum + ABS(Coefficients[Row, Column]);
1185         if Sum > ABS(Coefficients[Row, Row]) then
1186           Error := 1; { WARNING! convergence may not be }
1187                     { possible because matrix isn't }
1188                     { diagonally dominant }
1189         if ABS(Coefficients[Row, Row]) < TNNearlyZero then
1190           Error := 6; { Singular matrix - can't be solved }
1191                     { by the Gauss-Seidel method. }
1192       end; { while }
1193     end; { procedure TestForDiagDominance }
1194
1195 procedure MakeInitialGuess(Dimen      : integer;
1196                           var Coefficients : TNmatrix;
1197                           var Constants  : TNvector;
1198                           var Guess     : TNvector);
1199
1200 {-----}
1201 {- Input: Dimen, Coefficients, Constants -}
1202 {- Output: Guess -}
1203 {-}
1204 {- This procedure creates an initial approximation to the solution -}
1205 {- by dividing the Constants terms by the corresponding terms -}
1206 {- on the main diagonal of the Coefficients matrix. -}
1207 {-----}
1208
1209 var
1210   Term : integer;

```

```

1211
1212 begin
1213   FillChar(Guess, SizeOf(Guess), 0);
1214   for Term := 1 to Dimen do
1215     if ABS(Coefficients[Term, Term]) > TNNearlyZero then
1216       Guess[Term] := Constants[Term] / Coefficients[Term, Term];
1217 end; { procedure MakeInitialGuess }
1218
1219 procedure TestForConvergence(Dimen      : integer;
1220                             var OldApprox : TNvector;
1221                             var NewApprox : TNvector;
1222                             Tol          : Float;
1223                             var Done     : boolean;
1224                             var Product  : Float;
1225                             var Error    : byte);
1226
1227 {-----}
1228 {- Input: Dimen, OldApprox, NewApprox, Tol, Product -}
1229 {- Output: Done, Product, Error -}
1230 {-
1231 {- This procedure determines if the sequence of approximations -}
1232 {- has converged. For convergence to occur, the relative difference -}
1233 {- between each Term of OldApprox and NewApprox must be less than -}
1234 {- the tolerance, Tol. If so, Done = TRUE is returned. -}
1235 {-
1236 {- This procedure also determines if the sequence of approximations -}
1237 {- is diverging. Product records the total fractional change from -}
1238 {- the initial guess to the current iteration. If Product is greater -}
1239 {- than 1E20, then the sequence is assumed to have diverged. If so, -}
1240 {- Error = 7 is returned. -}
1241 {-----}
1242
1243 var
1244   Term : integer;
1245   PartProd : Float;
1246
1247 begin
1248   Done := true;
1249   PartProd := 0;
1250   for Term := 1 to Dimen do
1251     begin
1252       if ABS(OldApprox[Term] - NewApprox[Term]) > ABS(NewApprox[Term] * Tol) then
1253         Done := false;
1254       if (ABS(OldApprox[Term]) > TNNearlyZero) and (Error = 1) then
1255         { This is part of the divergence test }
1256         PartProd := PartProd + ABS(NewApprox[Term] / OldApprox[Term]);
1257     end;
1258     Product := Product * PartProd / Dimen;
1259     if Product > 1E20 then
1260       Error := 7 { Sequence is diverging }
1261 end; { procedure TestForConvergence }
1262
1263 procedure Iterate(Dimen      : integer;
1264                  var Coefficients : TNmatrix;
1265                  var Constants   : TNvector;
1266                  var Guess       : TNvector;
1267                  Tol             : Float;
1268                  MaxIter        : integer;
1269                  var Solution    : TNvector;
1270                  var Iter        : integer;
1271                  var Error       : byte);
1272
1273 {-----}
1274 {- Input: Dimen, Coefficients, Constants, Guess, Tol, MaxIter -}
1275 {- Output: Solution, Iter, Error -}
1276 {-
1277 {- This procedure performs the Gauss-Seidel iteration and -}
1278 {- returns either an error or the approximated solution and -}
1279 {- the number of iterations. -}
1280 {-----}
1281
1282 var
1283   Done : boolean;
1284   OldApprox, NewApprox : TNvector;
1285   Term, Loop : integer;
1286   FirstSum, SecondSum, Product : Float;
1287
1288 begin { procedure Iterate }
1289   Product := 1;
1290   Done := false;
1291   Iter := 0;
1292   NewApprox := Guess;
1293   OldApprox := Guess;
1294   while (Iter < MaxIter) and not(Done) and (Error <= 1) do
1295     begin
1296       Iter := Succ(Iter);
1297       for Term := 1 to Dimen do
1298         begin
1299           FirstSum := 0;
1300           SecondSum := 0;
1301           for Loop := 1 to Term - 1 do
1302             FirstSum := FirstSum + Coefficients[Term, Loop] * NewApprox[Loop];
1303           for Loop := Term + 1 to Dimen do
1304             SecondSum := SecondSum + Coefficients[Term, Loop] * OldApprox[Loop];
1305           NewApprox[Term] := (Constants[Term] - FirstSum - SecondSum) /
1306                             Coefficients[Term, Term];
1307         end;
1308       TestForConvergence(Dimen, OldApprox, NewApprox, Tol, Done, Product, Error);
1309       OldApprox := NewApprox;
1310     end; { while }
1311     if (Iter < MaxIter) and (Error = 1) then
1312       Error := 0; { The sequence converged, }
1313       { disregard the warning }
1314     if (Iter >= MaxIter) and (Error = 1) then
1315       Error := 1; { Matrix is not diagonally dominant; }
1316       { convergence is probably impossible }
1317     if (Iter >= MaxIter) and (Error = 0) then
1318       Error := 2; { Convergence IS possible; }
1319       { more iterations are needed }
1320   Solution := NewApprox;

```

Page 13, listing of MATRIX.INC, date is 18-02-93, file date is 01-01-80, size is 49712 bytes.

```
1321 end; { procedure Iterate }
1322
1323 begin { procedure Gauss_Seidel }
1324   TestInput(Dimen, Tol, MaxIter, Coefficients, Constants, Solution, Error);
1325   if Dimen > 1 then
1326     begin
1327       TestForDiagDominance(Dimen, Coefficients, Error);
1328       if Error < 2 then
1329         begin
1330           MakeInitialGuess(Dimen, Coefficients, Constants, Guess);
1331           Iterate(Dimen, Coefficients, Constants, Guess, Tol,
1332                 MaxIter, Solution, Iter, Error);
1333         end;
1334       end;
1335 end; { procedure Gauss_Seidel }
```

```

1 Unit GPSEngine;
2 {*****}
3 { This unit provides procedures to be used by the GPS unit. The procedures
4   are taylor made for the GPSengine from Magnavox.}
5 {*****}
6
7
8 Interface
9
10 {$N+,E+}
11
12 Uses MIASglob, GPSglob, crt;
13
14
15 Procedure InitGPSrec( Var error: Boolean);
16 {*****}
17 {Initialise the GPS receiver. If something went wrong, error := True.
18   Input :-
19   Output:error}
20 {*****}
21
22
23 Procedure CollectGPSrec( Var GPSint: GPSinttype);
24 {*****}
25 {get characters from buffers. Synchronise with the delimiters 'LF' and '$'.
26   Check for correct header and process the information to the variable GPSint
27   Input :-
28   Output:GPSint}
29 {*****}
30
31
32 Procedure ExecGPSrecCommand( command: commandtype);
33 {*****}
34 { Convert a GPScommand, to a commandstring for the GPS Engine and send
35   the command using the IWRITECOM routine in COMM_.int. The commands
36   here start with 'GPS:'.
37   Input : command
38   Output: commandstring on comport}
39 {*****}
40
41
42 Procedure CloseGPSrec;
43 {*****}
44 {Make sure the GPS receiver is back to normal. Restore interruptvectors etc
45   Input :-
46   Output:-}
47 {*****}
48
49
50 Implementation
51
52
53 Uses User, Miscell, com_4;comdisc}
54
55
56 Const
57   LF      = #10;{ linefeed}
58
59 Var
60   port0,
61   port1   :      Byte; { contains comportnr for port0 and port1
62                     of the GPSengine}
63   x, y     :      Shortint; {counter}
64   two2power :      Array[ 0..55] Of Double;
65   sv_id    :      Byte; { contains the satellite identity number
66                     for ephemeris transmissions}
67   Tsv_id,
68   Valid_Tsv_id :      timetype;{Time that sv_id should be valid}
69   Tpr,
70   Valid_Tpr   :      timetype;{ time that a set is valid, for timeout}
71   old_user_ms :      Real; { remember old time of measurement}
72   svccount    :      Shortint;{ count the number of PR yet received}
73   tempGPSint  :      GPSinttype;{ internal var containing all info received}
74   completeinfo :      Byte; { contains info for transfer of completeinfo}
75                     { or partial info if available}
76                     { 0 for partial, 1 for complete}
77
78 {----- start procedure initgpsrec-----}
79 Procedure InitGPSrec( Var error: Boolean);
80
81 Var
82   title,
83   line   :      String;
84   code   :      Integer;
85   varname,
86   value  :      String;
87   setupfile :      Text;
88
89 Begin
90
91   { set default values}
92   port1 := 1; { Engine port 1 is com 1}
93   port0 := 2; { Engine port 0 is com 2}
94   error:= True;
95   OpenConfigRead( setupfile, MIAScfgrname);
96   Repeat
97     Readln( setupfile, title);
98   Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'GPSENGINE'));
99
100  If Not Eof( setupfile) { if there is more in file}
101  Then Repeat
102    Readln( setupfile, line); { get a line}
103    Convert( line, varname, value); { extract the variable name
104                                   and value}
105
106    If (varname = 'PORT0')
107    Then Val( value, port0, code);
108
109    If ( varname = 'PORT1')
110    Then Val( value, port1, code);

```

```

111      If ( varname = 'COMPLETEINFO')
112      Then Val( value, completeinfo, code);
113      Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
114          { repeat until end of file}
115      CloseConfig( setupfile);
116
117          { initialise comports for
118            communication with Engine}
119      Setupcomport( port0, Ord( B4800), 8, Ord( None), 1);
120      Setupcomport( port1, Ord( B9600), 8, Ord( None), 1);
121          { empty receive and trans-
122            mit buffers}
123      Emptybuffer( port0, True);
124      Emptybuffer( port1, True);
125          { set interrupt vectors}
126      Installint( port0);
127      Installint( port1);
128          { save old interrupt}
129          { vectors}
130      error:= False;
131  End;
132  {----- end procedure initgpsrec-----}
133  {----- start procedure CollectGPSrec-----}
134  Procedure CollectGPSrec( Var GPSint: GPSinttype);
135
136  {----- start included procedures CollectGPSrec-----}
137  Procedure Collectport0( Var GPSint: GPSinttype);
138  {*****}
139  { This procedure collects data from port0 of the GPS ENGINE receiver.
140    Input : data from receive buffer
141    Output: updated GPSint.}
142  {*****}
143
144  {----- start included procedures collectport0-----}
145  Procedure StatusReport( rec: String; Var GPSint: gpsinttype);
146  {*****}
147  { This procedure reads the number of satellites being tracked}
148  {
149    Input : string from GPSEngine
150    Output: number of satellites being tracked}
151  {*****}
152
153  Var
154      value :      String;
155      code  :      Integer;
156
157  Begin
158      value:= Copy( rec, 12, Length( rec) -11);{ select everything
159                                                { but the header}
160
161      For x:= 1 To 2 Do
162      Begin
163          While value[1] <> ',' Do
164              value:= Copy( value, 2, Length( value) -1);
165          value:= Copy( value, 2, Length( value) - 1);
166      End;

```

```

166      value:= Copy( value, 1, 1);
167      Val( value, GPSint.numofsat, code);
168
169      If code <> 0
170      Then GPSint.numofsat := -1;
171  End;
172  {----- end included procedures collectport0-----}
173
174  Var
175      rec,
176      header      :      String;
177      recnum,
178      code         :      Integer;
179      time         :      timetype;
180      part         :      String;
181
182  {-----Start collectport0-----}
183  Begin
184      GPSint:= tempGPSint;
185      { begin collectport0}
186
187      If (charsinbuff( port0) > 80)
188      Then Begin
189          { wait until string long}
190          { enough to be valid}
191          { get on char from buffer}
192          rec:= Getcharbuff( port0);
193          { if begin of recordstring}
194          If ( rec = '$')
195          Then Begin
196              { read chars from buffer}
197              Repeat
198                  rec:= rec + Getcharbuff( port0);
199              Until ( Pos( LF, rec) <> 0) Or
200                  ( charsinbuff( port0) = 0);
201              { until a line feed found}
202              { or no more chars in buff}
203              If ( charsinbuff( port0) = 0)
204              Then Exit;
205              { if no lf found, exit}
206
207              { first part of string must}
208              { be $PMVSG for Engine}
209              header:= Copy( rec, 1, 7);
210              If ( header = '$PMVXG,')
211              Then Begin
212                  { next part is recordnumber}
213                  gotoxy( 1, 15);
214                  write( rec);
215                  SaveEquipmentMessage( 'RG: ' + rec);
216                  End
217                  Else;
218                  { what if $PMVXG not found?}
219                  End
220                  Else While (( rec <> LF) And
221                          ( Charsinbuff( port0) > 0)) Do
222                      { if not begin of record}
223                      { read buffer until lf}
224                      rec:= getcharbuff( port0);
225                  End;
226                  tempGPSint:= GPSint;
227                  {end collectport0}
228              {-----end collectport0-----}

```

```

221 Procedure Collectport1( Var GPSint: GPSinttype);
222 {*****}
223 { This procedure collects data from port1 of the GPS ENGINE receiver.
224   Input : data from receive buffer
225   Output: updated GPSint.}
226 {*****}
227
228 Type
229   temptype      =      Array[1..24] Of Byte;
230
231 {----- start included procedures collectport1-----}
232 Procedure Conv_ASCII_2_Val( rec: String; Var temp: temptype;
233   Var error: Boolean);
234 {*****}
235 { This procedure convert a string with pairs of ascii characters
236   to byte values. This string should contain 24 pairs of ascii char-
237   acters with the 'values' '0'..'9' or 'A'..'F'. The first pair should
238   begin at position 6 of the string. When the data is erroneous, then
239   the error flag is set True.
240   Input : received string
241   Output: array with 24 bytes
242   error when flag}
243 {*****}
244
245 Var
246   x, y          :      Integer;
247   value         :      Byte;
248   dum           :      Char;
249   code          :      Integer;
250
251 Begin
252   error:= False;          { initialise errorflag}
253
254   For x:= 0 To 23 Do      { count 24 sets of }
255   Begin                  { characters}
256     value:=0;             { reset value}
257     temp[ x+1] := 0;      { reset almanac}
258
259     For y:= 1 DownTo 0 Do { convert two characters to}
260     Begin                { value}
261       dum:= rec[6 + x * 3 + (1 - y)]; { get digit starting at }
262       { position 6}
263       If Not ( dum in ['0'..'9', 'A'..'F'])
264         { digit is hexadecimal}
265       Then Begin
266         error:= True;
267         Exit;          { error in received record}
268       End;
269
270       Case dum Of
271         '0'..'9': Val( dum, value, code);
272         'A'..'F': value:= Ord( dum) - Ord( 'A') + 10;
273       End;
274
275       value:= Round(value * two2power[ y * 4]);{ first digit is 16's}

```

```

276   { second digit is 1's}
277   temp[ x+1] := temp[ x+1] + value;
278 End;
279   { complete line converted +}
280 End;      { stored}
281
282
283 Function Scale( temptarray: temptype; pointer, startbit,
284   nr_of_bits: Byte): Double;
285 {*****}
286 { The function takes one or more bytes from an array called tempalmanac
287   and converts it into a value. Pointer indicates the first byte from
288   the array to be used. Startbit indicates the number of the first bit
289   to be used. The MSB has number 0, increasing to the LSB's. Nr_of_bits
290   indicates the number of bits to be used.
291   WARNING: THIS FUNCTION WILL ONLY WORK A MAXIMUM OF 32 BITS, BECAUSE
292   OF THE ROUND FUNCTION.
293   Input : array with bytes
294   pointer for first byte
295   startbitnumber
296   number of bits
297   Output: Value of the number indicated with
298   the input variables}
299 {*****}
300
301 Var
302   temp           :      Double;
303   temppointer    :      Byte;
304   x              :      Byte;
305   leftoverbits   :      Byte;
306
307 Begin
308   temppointer:= pointer; { save the pointer in the array here}
309   temp:= temptarray[ pointer];
310   Inc( pointer);        { take the pointer'th byte from the array}
311   While (startbit + nr_of_bits) - ((pointer - temppointer) * 8) > 0 Do
312   Begin                  { repeat this, until enough bytes are taken}
313     temp:= temp * 256 + temptarray[ pointer];
314     Inc( pointer);      { give every byte its position ref weight}
315   End;
316
317   { Delete MSB's that should not be used}
318   { x is number of the bits that should not be used}
319   { subtract the values indicated by those bits}
320   If startbit > 0
321   Then For x:= 0 To startbit - 1 Do
322     If ( temp >= two2power[ (pointer - temppointer) * 8
323       - x - 1] )
324     Then temp:= temp - two2power[ (pointer - temppointer) * 8
325       - x - 1];
326
327   { delete LSB's that should not be used}
328   { x is number of LSB's that should not be used}
329   leftoverbits:= - (( startbit + nr_of_bits) -
330     ((pointer - temppointer) * 8));
331   While leftoverbits > 0 Do

```

```

331           { divide by 2, to correct for these LSB's}
332   Begin           { as many times as there are LSB's too much}
333       Dec( leftoverbits);
334       temp:= temp / 2;
335       temp:= Trunc( temp);
336   End;
337   Scale:= temp;
338 End;
339
340
341 Function Twoscomplement( number: Double; nr_of_bits: Byte): Double;
342 {*****}
343 {A number is converted to the decimal 'number' using the natural binary
344 code. This number was a twoscomplement number. This function decodes
345 'number' and encodes it again using the two's complement code.
346 If the MSB indicated by the nr_of_bits variable is 0, then the two's
347 complement number is the same as the binary number. If the MSB is 1,
348 then the two's complement number can be found by: - 2^nr_of_bits +
349 number.
350           Input : binary number
351                   number of bits
352           Output: two's complement of the number}
353 {*****}
354 Begin
355     If number > two2power[ nr_of_bits - 1]
356     Then Twoscomplement:= - two2power[ nr_of_bits] + number
357     Else Twoscomplement:= number;
358 End;
359
360
361 Procedure Rawdata( rec: String; Var GPSint: GPSinttype;
362                   Var svcount: Shortint);
363 {*****}
364 { This procedure converts the received string from the Engine, which
365 contains rawdata, to variables in the GPSint record.
366           Input : received string
367           Output: variables in GPSint
368                   indication on end of rawdata cycle}
369 {*****}
370
371 Const
372     c      =      299792458;      { speed of light}
373     l1freq=      1575420000.0;    { frequency of l1 carrier }
374
375 Var
376     user_ms,      { user time in GPS receiver
377                   (ms)}
378     chnl_ms,      { channel time in GPS
379                   receiver (ms)}
380     phi,          { integrated carrier phase
381                   l1 wavelengths}
382     phi_frac,     { idem, fractional. LSB=(l1
383                   wavelength)/256}
384     code          { raw code offset, l1 wave-
385                   lengths}
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440

```

```

441      ( code / l1freq);
442      rxtime:= user_time;
443      flag:= False;
444
445      End;
446 { This is made a comment, to make the program work better with a disc.
447 In that case, it may assume a wrong number of satellites being tracked
448 giving erroneous results. By not incrementing 'svcount', the errors can
449 be prevented. The complete cycle of measurements is noticed as soon as
450 the next cycle begins.)
451      Inc( svcount);
452      End
453 Else Begin
454      Restore_buffer( port1, LF+rec);{ put received string back
455                                     for later use}
456      old_user_ms:= user_ms;
457      svcount:= GPSint.numofsat;
458
459      Date_and_time( Tpr);
460      End;
461 End; { End of procedure rawdata}
462
463 Procedure Ionoscor( rec: String; Var GPSint: GPSinttype);
464 {*****}
465 { This procedure converts a received Engine string, containing Iono-
466 spheric information, to variables. It takes one, two or three bytes
467 from the tempionos, converts this to a value and adds an exponent.
468 The information used, can be found in Appendix 3 to Annex A to
469 STANAG 4249, 1 August 1990.
470      Input : received recordnumber
471             received string
472      Output: updated almanac in GPSint}
473 {*****}
474
475 Var
476      tempionos      :      temptype;
477      error          :      Boolean;
478
479 Begin
480      Conv_ASCII_2_val( rec, tempionos, error);
481      If error
482      Then Exit;
483      With GPSint.iono Do
484      Begin
485          alfa0:= Twoscomplement( Scale( tempionos, 2, 0, 8), 8);
486          alfa0:= alfa0 / two2power[ 30];
487
488          alfa1:= Twoscomplement( Scale( tempionos, 3, 0, 8), 8);
489          alfa1:= alfa1 / two2power[ 27];
490
491          alfa2:= Twoscomplement( Scale( tempionos, 4, 0, 8), 8);
492          alfa2:= alfa2 / two2power[ 24];
493
494          alfa3:= Twoscomplement( Scale( tempionos, 5, 0, 8), 8);
495
496          alfa3:= alfa3 / two2power[ 24];
497
498          beta0:= Twoscomplement( Scale( tempionos, 6, 0, 8), 8);
499          beta0:= beta0 * two2power[ 11];
500
501          beta1:= Twoscomplement( Scale( tempionos, 7, 0, 8), 8);
502          beta1:= beta1 * two2power[ 14];
503
504          beta2:= Twoscomplement( Scale( tempionos, 8, 0, 8), 8);
505          beta2:= beta2 * two2power[ 16];
506
507          beta3:= Twoscomplement( Scale( tempionos, 9, 0, 8), 8);
508          beta3:= beta3 * two2power[ 16];
509      End; { End of with}
510      Date_and_time( GPSint.Tionos);
511 End; { End of procedure Ionoscor}
512
513
514 Procedure Get_SV_id( rec: String; Var sv_id: Byte);
515 {*****}
516 { This procedure is needed, because the gpsengine sends the ephemeris
517 in several records. Only the first record contains the SV-id of the
518 SV from which the ephemeris originates
519      Input : received string
520      Output: satellite identity}
521 {*****}
522
523 Var
524      subrec      :      String;
525      code        :      Integer;
526
527 Begin
528      subrec:= Copy( rec, 6, 3);      { sv-id, position unknown}
529      Val( subrec, sv_id, code);      { convert string to value}
530      If ( code <> 0)                  { on error clear sv_id}
531      Then sv_id:=0;
532
533      Date_and_time( Tsv_id);
534 End;
535
536
537 Procedure Clockinfo( rec: String; sv_id: Byte;
538                     Var GPSint: GPSinttype);
539 {*****}
540 { This procedure converts received Engine strings, containing Clock-
541 information per satellite, to variables. It takes one, two or three
542 bytes from the tempclock, converts this to a value and adds an
543 exponent. The information used, can be found in Appendix 3 to Annex
544 A to STANAG 4249, 1 August 1990.
545      Input : received string
546             satellite identity
547      Output: updated clock info per SV in GPSint}
548 {*****}
549
550 Var
551      tempclock      :      temptype;

```



```

551      error      :      Boolean;
552      sumtime,
553      time       :      timetype;
554      dum        :      Double;
555
556  Begin
557      With GPSint Do
558      Begin
559          Date_and_Time( time);          { get current system time}
560          AddTime( Tsv_id, Valid_Tsv_id, sumtime);
561          If (sv_id = 0) Or Later( time, sumtime)
562              { if Satellite identity
563              number is 0, or the infor-
564              mation is timed out, than
565              no valid information}
566          Then Begin
567              ErrorTime( Tsv_id);
568              Exit;
569          End;
570      End;
571
572      Conv_ASCII_2_val( rec, tempclock, error);
573      If error
574      Then Exit;
575
576      With GPSint.prn[ sv_id].clock Do
577      Begin
578          Tgd:= Twoscomplement( Scale( tempclock, 15, 0, 8), 8);
579          Tgd:= Tgd / two2power[ 31];
580
581          toc:= Scale( tempclock, 17, 0, 16);
582          toc:= Round( toc * two2power[ 4]);
583          If toc > 604784          { check on range}
584          Then Begin
585              ErrorTime( GPSint.prn[ sv_id].Tck);
586              Exit;
587          End;
588
589          af2:= Twoscomplement( Scale( tempclock, 19, 0, 8), 8);
590          af2:= af2 / two2power[ 55];
591
592          af1:= Twoscomplement( Scale( tempclock, 20, 0, 16), 16);
593          af1:= af1 / two2power[ 43];
594
595          af0:= Twoscomplement( Scale( tempclock, 22, 0, 22), 22);
596          af0:= af0 / two2power[ 31];
597
598          dum:= Scale( tempclock, 3, 6, 2) * 256;
599          IODC:= Integer( Round(dum));
600          dum:= Scale( tempclock, 16, 0, 8);
601          IODC:= IODC + Integer( Round(dum));
602      End;      { End with}
603      With GPSint.prn[ sv_id] Do
604      Begin
605          health:= Round( Scale( tempclock, 3, 0, 6));
606
607          { set the 6 LSB's of
608          health by health from
609          SV record}
610          If Round( Scale( tempclock, 3, 0, 1)) = 0
611          Then health:= ( health And $1F) { healthy force zero's}
612          Else health:= ( health Or $E0); { unhealthy force one's}
613          { use MSB from SV record
614          to set health to 'alldata
615          bad' or 'all data ok'}
616
617          Date_and_Time( Tck);
618          End;      { End with}
619      End;      { End procedure clockinfo}
620
621  Procedure Ephemeris( recnum: Integer; rec: String; sv_id: Byte;
622                      Var GPSint: GPSinttype);
623  {*****}
624  { This procedure converts received Engine strings, containing Ephem-
625  eris information, to variables. It takes one, two or three bytes
626  from the tempephem, converts this to a value and adds an exponent.
627  The information used, can be found in Appendix 3 to Annex A to
628  STANAG 4249, 1 August 1990.
629
630      Input : received recordnumber
631             received string
632             satellite identity
633      Output: updated ephemeris in GPSint}
634  {*****}
635  Var
636      tempephem      :      temptype;
637      error          :      Boolean;
638      time,
639      sumtime        :      timetype;
640
641  Begin
642      Date_and_Time( time);          { get current system time}
643      AddTime( Tsv_id, Valid_Tsv_id, sumtime);
644      If (sv_id = 0) Or Later( time, sumtime)
645          { if Satellite identity
646          number is 0, or the infor-
647          mation is timed out, than
648          no valid information}
649      Then Begin
650          ErrorTime( Tsv_id);
651          Exit;
652      End;
653
654      Conv_ASCII_2_val( rec, tempephem, error);
655      If error
656      Then Exit;
657
658      With GPSint.prn[ sv_id].ephemeris Do
659      Begin
660          Case recnum Of
661              202: Begin
662                  IODE:= tempephem[ 1];

```

```

661
662      Crs:= Twoscomplement( Scale( tempephem, 2, 0, 16), 16);
663      Crs:= Crs / two2power[ 5];
664
665      deltan:= Twoscomplement( Scale( tempephem, 4, 0, 16), 16);
666      deltan:= deltan / two2power[ 43];
667
668      Mo:= Twoscomplement( Scale( tempephem, 6, 0, 32), 32);
669      Mo:= Mo / two2power[ 31];
670
671      Cuc:= Twoscomplement( Scale( tempephem, 10, 0, 16), 16);
672      Cuc:= Cuc / two2power[ 29];
673
674      e:= Scale( tempephem, 12, 0, 32);
675      e:= e / two2power[ 33];
676      If e > 0.03          { Check on range}
677      Then Begin
678          ErrorTime( GPSint.prn[ sv_id].Tephem);
679          Exit;
680      End;
681
682      Cus:= Twoscomplement( Scale( tempephem, 16, 0, 16), 16);
683      Cus:= Cus / two2power[ 29];
684
685      Asqrt:= Scale( tempephem, 18, 0, 32);
686      Asqrt:= Asqrt / two2power[ 19];
687
688      toe:= Scale( tempephem, 22, 0, 16);
689      toe:= toe * two2power[ 4];
690      If toe > 604784      { Check on range}
691      Then Begin
692          ErrorTime( GPSint.prn[ sv_id].Tephem);
693          Exit;
694      End;
695
696      Date_and_Time( GPSint.prn[ sv_id].Tephem);
697      End; { End of Case 202}
698 203: Begin
699      If (tempephem[22] <> IODE) {new ephemeris being}
700      Then Begin {uploaded. Subframe 2}
701          ErrorTime( GPSint.prn[ sv_id].Tephem);
702          Exit {ready, subframe 3 not}
703      End; {don't use this sv now}
704
705      Cic:= Twoscomplement( Scale( tempephem, 1, 0, 16), 16);
706      Cic:= Cic / two2power[ 29];
707
708      omegao:= Twoscomplement( Scale( tempephem, 3, 0, 32), 32);
709      omegao:= omegao / two2power[ 31];
710
711      Cis:= Twoscomplement( Scale( tempephem, 7, 0, 16), 16);
712      Cis:= Cis / two2power[ 29];
713
714      io:= Twoscomplement( Scale( tempephem, 9, 0, 32), 32);
715      io:= io / two2power[ 31];
716
717      Crc:= Twoscomplement( Scale( tempephem, 13, 0, 16), 16);
718      Crc:= Crc / two2power[ 5];
719
720      omega:= Twoscomplement( Scale( tempephem, 15, 0, 32),
721                               32);
722      omega:= omega / two2power[ 31];
723
724      omegadot:= Twoscomplement( Scale( tempephem, 19, 0, 24),
725                                 24);
726      omegadot:= omegadot / two2power[ 43];
727
728      IDOT:= Twoscomplement( Scale( tempephem, 23, 0, 14), 14);
729      IDOT:= IDOT / two2power[ 43];
730
731      Date_and_Time( GPSint.prn[ sv_id].Tephem);
732      sv_id:= 0; { When last record for
733                { satellite nr: sv_id is
734                { used, then reset sv_id
735                { so, no errors are made
736                { if records 201..203 are
737                { received without record
738                { 200( correct sv_id))
739
740      End; { End of Case 203}
741      End; { End of case}
742      End; { End of with}
743      End; { End of procedure Ephemeris}
744 {----- end included procedures collectport1-----}
745 Var
746     rec,
747     dumstr      :      String;
748     dumch       :      Char;
749     recnum,
750     code        :      Integer;
751     dum         :      Integer;
752     x           :      Byte;
753     time,
754     result      :      timetype;
755     part        :      String;
756 {----- start procedure collect port1-----}
757 Begin
758     For x:= 1 To 32 Do
759         gpsint.prn[x].flag:= true;
760
761     If (Charsinbuff( port1) > 80) { only start if there}
762     Then Begin { are enough chars in buff}
763         rec:= Getcharbuff( port1); { Get first char}
764
765         If ( rec = LF) { If this is a line feed}
766         Then Begin { then we expect a new}
767             rec:=''; { leave a LF in buffer}
768             Repeat { line to start}
769                 dumch:= LookBuff( port1);
770

```

```

771         If ( dumch <> LF)
772         Then rec:= rec + Getcharbuff( port1);
773     Until ( dumch = LF) Or ( charsinbuff( port1)= 0);
774         { continue until end of
775         line is detected, or
776         no more chars}
777         { if no more chars, then
778         error in received line}
779     If ( charsinbuff( port1) = 0)
780     Then Exit;
781     If ( rec[4] = ' ') { for right alignment of
782     record number}
783     Then rec:= ' '+ rec; { insert space if necessary}
784
785         { copy the number}
786     dumstr:= Copy( rec, 1, 4);
787         { convert string to value}
788     Val( dumstr, recnum, code);
789     If ( code = 0) { if no error converting}
790     Then Begin
791 gotoxy( 1, 17);
792 write( rec);
793 SaveEquipmentMessage( 'RG: ' + rec);
794         Case recnum Of
795             1 : Rawdata( rec, GPSint, svcount);
796         End; { end case}
797     End; { end begin}
798     Else ; { what to do if error?}
799     End
800 Else Begin { if no begin of line}
801     Repeat { skip chars from buffer}
802         dumch:= LookBuff( port1);
803         If ( dumch <> LF)
804         Then dumch:= Getcharbuff( port1);
805     Until ( dumch = LF) Or ( charsinbuff( port1) = 0);
806         { a lf was found or the
807         buffer is empty}
808     End;
809 End; { end if charsinbuff}
810
811 End; { end collectport1}
812 {----- end collectport1-----}
813 {----- end included procedures CollectGPSrec-----}
814
815 Var
816     counter : Integer;
817 {----- start procedure CollectGPSrec-----}
818 Begin
819     counter:= 1; { now we read 2 message}
820     Repeat { from the engine receiver}
821         Collectport0( GPSint); { these contain most certain}
822         Collectport1( GPSint); { ly all the measurements}
823         Dec( counter);
824     Until (counter = 0);
825 End;

```

```

826 {----- end procedure CollectGPSrec-----}
827
828
829 {----- start procedure execgpsrecommand-----}
830 Procedure ExecGPSrecCommand( command: commandtype);
831     { IN THE COMMENT, UNDER-
832     SCORES ' ' SHOULD BE
833     READ AS SPACES ' '}
834 Var
835     substr,
836     sendline : String;
837     code,
838     sv : Integer;
839     rate : Integer;
840
841 Begin
842     sendline:= '';
843
844     substr:= Copy( command, 5, Length( command) - 4);
845     { delete header: 'GPS:'}
846     If substr = 'RESET'
847     Then Begin
848         sendline:= '$PMVXG,018,T'#13#10; { RESTART THE ENGINE WITH
849         TEPID START}
850     { IWriteCom( port0, sendline);
851     SaveEquipmentMessage( 'SG: ' + sendline);
852     } End; { clear port0 outputlist}
853
854
855     If (Copy( substr, 1, 4) = 'INIT') And
856     ( Length( substr) >= 52)
857     Then Begin
858         sendline:= '$PMVXG,000,'+
859         Copy( substr, 6, 2) + ',' +
860         Copy( substr, 9, 2) + ',' + { INITIALISES THE ENGINE}
861         '19' +
862         Copy( substr, 12, 2) + ',' + { RECEIVER WITH POSITION}
863         Copy( substr, 15, 6) + ',' + { ETC. FORMAT IS: 'INIT_'}
864         Copy( substr, 22, 9) + ',' + { DD_MM_YY_HHMM_DDMM.MMMM}
865         Copy( substr, 32, 1) + ',' + { _N_DDDMM.MMMM_E_HHHHH.H_)
866         { AA.A_EL'}
867         Copy( substr, 34, 10)+ ',' + { WHERE DD_MM_YY IS THE}
868         Copy( substr, 45, 1) + ',' + { DATE IN DAY, MONTH, AND}
869         Copy( substr, 47, 7) + ',' +
870         #13#10;
871
872         IWriteCom( port0, sendline);
873     SaveEquipmentMessage( 'SG: ' + sendline);
874
875
876     sendline:=
877     '$PMVXG,001,,,' + { YEAR. HHMM IS TIME, }
878     Copy( substr, 55, 4) + ',' + { DDMM.MMMM IS LATITUDE}
879     ',,, ' + { IN DEGREES AND MINUTES,}
880     Copy( substr, 60, 2) + ',' + { N IS NORTH OR SOUTH }

```

```

881      ', '+#13#10;          { (N/S), DDDMM.MMMM IS }
882                          { LONGITUDE IN DEGREES }
883                          { AND MINUTES, E IS EAST}
884                          { OR WEST (E/W) AND }
885                          { HHHHH.H IS ALTITUDE }
886                          { ABOVE MEAN SEA LEVEL,}
887                          { AA.A IS HORIZONTAL}
888                          { ACCELERATION FACTOR}
889                          { EL IS ELEVATION LIMIT}
890      IWriteCom( port0, sendline);
891      SaveEquipmentMessage( 'SG: ' + sendline);
892
893      sendline:= '$PMVXG,024,-,+,-,-,-'#+13#10;
894      IWriteCom( port0, sendline);
895      SaveEquipmentMessage( 'SG: ' + sendline);
896
897      { NO NAV RESULT, RAW
898      MEASUREMENTS, APMANAC &
899      EPHEMERIS, NO CONTROL
900      INFO, NO TIME RECOVERY,
901      NO FULL DEBUG, NO PARTIAL
902      DEBUG}
903      sendline:= '$PMVXG,023,D,G,,,,'#+13#10;
904      IWriteCom( port0, sendline);
905      SaveEquipmentMessage( 'SG: ' + sendline);
906
907      { SET TIME RECOVERY ON}
908      sendline:= '$PMVXG,007,,1,,,,,'#+13#10;
909      IWriteCom( port0, sendline);          { NO OUTPUT ON CTRL PORT}
910      SaveEquipmentMessage( 'SG: ' + sendline);
911      sendline:= '$PMVXG,007,000,0,1,,1,,,,'#+13#10;
912      IWriteCom( port0, sendline);          { OUTPUT MESSAGE 000}
913      SaveEquipmentMessage( 'SG: ' + sendline);
914      sendline:= '$PMVXG,007,001,0,1,,1,,,,'#+13#10;
915      IWriteCom( port0, sendline);          { OUTPUT MESSAGE 001}
916      SaveEquipmentMessage( 'SG: ' + sendline);
917      sendline:= '$PMVXG,007,021,0,1,,1,,,,'#+13#10;
918      IWriteCom( port0, sendline);          { OUTPUT MESSAGE 021}
919      SaveEquipmentMessage( 'SG: ' + sendline);
920
921      End;
922      If substr = 'SEND EPHEMERIS ETC'
923      Then Begin
924          sendline:= '$PMVXG,027,,,,2,2'#+13#10;
925          IWriteCom( port0, sendline);
926          SaveEquipmentMessage( 'SG: ' + sendline);
927          End;
928          { OUTPUT EPHEMERIS AND
929          ALMANAC NOW}
930      End;
931
932      {----- End procedure execgpsrecommand-----}
933
934      {----- start procedure closegpsrec-----}
935      Procedure CloseGPSrec;
936
937      Var
938          setupfile      :      Text;
939          value           :      String;
940      Begin
941          Removeint( port0);
942          Removeint( port1);
943
944          OpenConfigWrite( setupfile, MIAScfigname);
945          Writeln( setupfile, 'GPSENGINE');
946
947          Str( port0, value);
948          Writeln( setupfile, #9'port0 = ', value, ';');
949
950          Str( port1, value);
951          Writeln( setupfile, #9'port1 = ', value, ';');
952
953          Str( completeinfo, value);
954          Writeln( setupfile, #9'completeinfo = ', value, ';');
955          CloseConfig( setupfile);
956      End;
957      {----- end procedure closegpsrec-----}
958
959
960      {----- start initialising -----}
961      Begin
962          { initialising part}
963          { used to make a table}
964          { containing the power of 2}
965          { fill most left position}
966          { calculate 2^1 to 2^55}
967          two2power[ 0]:= 1;
968          For x:= 1 To 55 Do
969              two2power[ x]:= two2power[ x-1] * 2;
970
971          sv_id:=0;
972          { set sv_id to 0, which is
973          invalid, so no mistake can
974          be made}
975
976          ErrorTime( Tsv_id);
977          ErrorTime( Valid_Tsv_id);
978          Valid_Tsv_id.minute:= 5;
979          { the sv_id stays valid for 5}
980          { minutes ( see record 200)}
981
982          ErrorTime( Tpr);
983          ErrorTime( Valid_Tpr);
984          Valid_Tpr.sec:= 1;
985          { the pr are maximum valid for}
986          { 1 second, this is the update}
987          { rate of the GPSEngine rx}
988
989          old_user_ms:= 0;
990          svcoun:= 0;
991
992          With tempGPSint Do
993              Begin
994                  flag:= True;
995                  ErrorTime( Tionos);
996                  For x:= 1 To 32 Do
997                      With prn[x] Do
998                          Begin
999                              flag:= True;

```

Page 10, listing of GPSENGIN.PAS, date is 18-02-93, file date is 17-02-93, size is 45690 bytes.

```
991         ErrorTime( Tck);
992         ErrorTime( Tephem);
993     End;
994     numofsat:= 0;
995 End;
996
997     completeinfo:= 1;
998 End.
999 {----- end initialising -----}
1000 {----- end Unit GPSENGINE -----}
```

```

1 Unit MLSbendix;
2 {*****}
3 { This unit is meant to be used with a Bendix MLS-20A receiver. This
4 receiver is a quasi MLSarinc727 receiver. The mls data words are not
5 passed through by this receiver. The basic datawords are simulated by
6 assigning variables with static basic data values}
7 {*****}
8
9 {$N+,E+}
10
11 Interface
12
13
14 Uses MIASglob, MLScglob, crt;
15
16
17 Procedure InitMLSrec( Var error: Boolean);
18 {*****}
19 { Initialise the Arinc 727 receiver. If something went wrong, error := True.
20   Input : -
21   Output: error}
22 {*****}
23
24
25 Procedure CollectMLSrec( Var MLSint: MLSinttype);
26 {*****}
27 { Get Arinc 429 words; Skip the words that are not necessary; check the
28 necessary words on parity. ADW's are checked on CRC.
29   Input : -
30   Output: MLSint}
31 {*****}
32
33
34 Procedure ExecMLSrecCommand( command: commandtype);
35 {*****}
36 { Convert a MLSccommand, to a command word for the ARinc 727 MLS receiver.
37 That is, a Arinc 429 word. Then send the word using the Arinc 429 tx
38 channel.
39   Input : command
40   Output: Arinc 429 word on 429 tx port}
41 {*****}
42
43
44 Procedure CloseMLSrec;
45 {*****}
46 { Make sure the ARINC 727 MLS receiver is back to normal. Restore interrupt-
47 vectors etc.
48   Input : -
49   Output: -}
50 {*****}
51
52
53 Implementation
54
55 Uses Ar429comm, Ar429, Miscell, User, ADW;
56
57 Type
58   ADW_prestype = Array[1..4] Of Boolean;
59   ByteArray = Array[1..10] Of Byte;
60
61 Const
62   mlsfreq_lab= 036; { mlsfrequency label}
63   azimuth_lab = 151; { azimuth function label}
64   elevat_lab = 152; { elevation function label}
65   BackAz_lab = 240; { backazimuth function label}
66
67 Var
68   ADW_A_pres,
69   ADW_B_pres,
70   ADW_C_pres : ADW_prestype;
71   ADW_A,
72   ADW_B,
73   ADW_C : ADWtype;
74   x : Integer;
75   AngleBegin : Boolean;
76   tempMLSint : MLSinttype;
77   Tangle,
78   Valid_Tangle : timetype;
79   completeinfo : Byte;
80   irq : Byte;
81   cardaddress : Word;
82   badwline : String;
83
84 {-----Start InitMLSrec-----}
85 Procedure InitMLSrec( Var error: Boolean);
86
87 Var
88   title,
89   line : String;
90   code : Integer;
91   varname,
92   value : String;
93   setupfile : Text;
94   selecttable : arraytype;
95
96 Begin
97   error:= True;
98   cardaddress:= $280;
99   irq:= 7;
100  badwline:= '';
101
102  OpenConfigRead( setupfile, MIAScfname);
103  Repeat { find MLSBENDIX part of
104                                     config file}
105      ReadLn( setupfile, title);
106  Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'MLSBENDIX'));
107
108  If Not Eof( setupfile) { if there is more in file}
109  Then Repeat
110      ReadLn( setupfile, line); { get a line}

```

```

111      Convert( line, varname, value); { extract the variable name
112                                     and value}
113      If ( varname = 'COMPLETEINFO')
114      Then Val( value, completeinfo, code);
115
116      If ( varname = 'CARDADDRESS')
117      Then Val( value, cardaddress, code);
118
119      If ( varname = 'IRQ')
120      Then Val( value, irq, code);
121      Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
122      { repeat until end of file}
123      CloseConfig( setupfile);
124
125      Ar429comm.InitAr429;           { initialise comm-port for}
126                                     { VHF link}
127      selecttable[ 0]:= mlsfreq_lab; { initialise ARINC card}
128      selecttable[ 1]:= azimuth_lab; { by filling in a label-}
129      selecttable[ 2]:= elevat_lab;  { select table}
130      selecttable[ 3]:= backaz_lab;
131
132      Ar429.InitAr429( error, selecttable, 4);
133
134      InitKaartadres( cardaddress); { Initialise ADW-card}
135      KiesIRQ( IRQ);
136      Install_ADW_int;
137      ProgTrigFunktie( 1, $0A);      { Select BDW 1}
138      ProgTrigFunktie( 2, $1F);      { Select BDW 2}
139      ProgTrigFunktie( 3, $55);
140      ProgTrigFunktie( 4, $11);
141      ProgTrigFunktie( 5, $1B);
142      ProgTrigFunktie( 6, $58);      { Select BDW 6}
143      ProgTrigFunktie( 7, $27);      { Select ADW A}
144      End;
145      {-----End InitMLSrec-----}
146
147
148      {-----Start CollectMLSrec-----}
149      Procedure CollectMLSrec( Var MLSint: MLSinttype);
150
151      {----- start included procedures CollectMLSrec-----}
152      Function ADW_present( ADW_pres: ADW_prestype): Boolean;
153      {*****}
154      { Check if all 4 arinc429 words, that form an adw, are present. A
155      true is output if all 4 are present, if not, a false is output.
156      Input : Array with present flags
157      Output: Boolean.}
158      {*****}
159
160      Begin
161          ADW_present:= ADW_pres[ 1] And ADW_pres[ 2] And
162                      ADW_pres[ 3] And ADW_pres[ 4];
163      End;
164
165

```

```

166      Function Hamming_Fail( ADW: ADWtype): Boolean;
167      {*****}
168      { This function checks the hammingcode in the specified ADW. If the
169      hammingcode was correct, then the output is true. See Annex 10 p 1508
170      Input : ADW
171      Output: Boolean}
172      { Author: Maarten Uit de Haag;
173      Revised: Marco Meijer}
174      {*****}
175      Var
176          x          : Integer;
177          succes      : Boolean;
178          check       : Array[ 0..6] Of Integer;
179
180      Begin
181          succes:= True;
182
183          For x:= 0 To 4 Do
184              Begin
185                  check[ x]:= ADW[13 + x] + ADW[14 + x] + ADW[15 + x] +
186                             ADW[16 + x] + ADW[17 + x] + ADW[18 + x] +
187                             ADW[20 + x] + ADW[22 + x] + ADW[24 + x] +
188                             ADW[25 + x] + ADW[28 + x] + ADW[29 + x] +
189                             ADW[31 + x] + ADW[32 + x] + ADW[33 + x] +
190                             ADW[35 + x] + ADW[36 + x] + ADW[38 + x] +
191                             ADW[41 + x] + ADW[44 + x] + ADW[45 + x] +
192                             ADW[46 + x] + ADW[50 + x] + ADW[52 + x] +
193                             ADW[53 + x] + ADW[54 + x] + ADW[55 + x] +
194                             ADW[58 + x] + ADW[60 + x] + ADW[64 + x] +
195                             ADW[65 + x];
196                  check[ x]:= check[ x] Mod 2;
197                  succes:= succes And ( check[ x] = ADW[ 70 + x]);
198              End;
199              check[ 5]:= ADW[13 ] + ADW[14 ] + ADW[15 ] + ADW[16 ] +
200                          ADW[17 ] + ADW[19 ] + ADW[21 ] + ADW[23 ] +
201                          ADW[24 ] + ADW[27 ] + ADW[28 ] + ADW[30 ] +
202                          ADW[31 ] + ADW[32 ] + ADW[34 ] + ADW[35 ] +
203                          ADW[37 ] + ADW[40 ] + ADW[43 ] + ADW[44 ] +
204                          ADW[45 ] + ADW[49 ] + ADW[51 ] + ADW[52 ] +
205                          ADW[53 ] + ADW[54 ] + ADW[57 ] + ADW[59 ] +
206                          ADW[63 ] + ADW[64 ] + ADW[69 ];
207              check[ 5]:= check[ 5] Mod 2;
208              succes:= succes And ( check[ 5] = ADW[ 75]);
209
210              check[ 6]:= 0;
211              For x:= 13 To 75 Do
212                  check[ 6]:= check[ 6] + ADW[ x];
213              check[ 6]:= check[ 6] Mod 2;
214              succes:= succes And ( check[ 6] = ADW[ 76]);
215
216              Hamming_Fail:= Not Succes;
217      End;
218
219
220      Function Adress_Fail( ADW: ADWtype): Boolean;

```

```

221 {*****}
222 { This function checks the parity of the address in the specified ADW.
223   If the address was correct, then the output is False. See Annex 10 p 150
224     Input : ADW
225     Output: Boolean}
226 { Author: Maarten Uit de Haag;
227   Revised: Marco Meijer}
228 {*****}
229 Var
230   check      :      Array[ 1..2] Of Integer;
231
232 Begin
233   check[ 1]:= ADW[ 13] + ADW[ 14] + ADW[ 15] + ADW[ 16] + ADW[ 17] +
234     ADW[ 18];
235   check[ 1]:= check[ 1] Mod 2;
236   check[ 2]:= ADW[ 14] + ADW[ 16] + ADW[ 18];
237   check[ 2]:= check[ 2] Mod 2;
238
239   Adress_Fail:= Not ((check[ 1] = ADW[ 19]) And
240     ( check[ 2] = ADW[ 20]));
241 End;
242
243
244 Function ADW_adress( ADW: ADWtype): Byte;
245 {*****}
246 { This function takes bits 13 to 18 of the ADW and translates these
247   bits to an adress. This is a normal binary code. See Annex 10 p 150
248     Input : ADW
249     Output: adress}
250 {*****}
251 Var
252   adress,
253   mult,
254   x      :      Integer;
255
256
257 Begin
258   adress:= 0;
259   mult:= 32;
260   For x:= 0 To 5 Do
261     Begin
262       adress:= adress + ADW[ 13 + x] * mult;
263       mult:= mult Div 2;
264     End;
265   ADW_adress:= adress;
266 End;
267
268
269 Function Conv_ADW( ADW: ADWtype; start: Byte; number: Byte): Word;
270 {*****}
271 { This function converts single bits to a number. 'Start' indicates
272   the start bit in the ADW. First bit is bitnumber 1. 'Number' indicates
273   the number of bits to be used for the number to be formed. The number
274   is output as a Word. See Annex 10 p 60CC; LSB first
275     Input : ADW

```

```

276     startbit
277     number of bits
278     Output: number}
279 {*****}
280 Var
281   value,
282   mult      :      Word;
283
284 Begin
285   value:= 0;
286   mult:= 1;
287   For x:= 0 To number Do
288     Begin
289       value:= value + ADW[ start + x] * mult;
290       mult:= mult * 2;
291     End;
292   Conv_ADW:= value;
293 End;
294
295
296
297 Procedure ADW_A_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype;
298   a_label: byte; Var ADW_A_pres: ADW_prestype;
299   Var ADW_A: ADWtype);
300 {*****}
301 { This procedure converts the A-ADW's to Pascal variables.
302   See Annex 10 p 150A etc.
303     Input : Arinc 429 word
304           Arinc 429 label
305           ADW_A_present array of boolean
306     Output: Mlsint
307           A-ADW }
308 {*****}
309 Var
310   adress      :      Byte;
311
312
313 Begin
314   Case a_label Of
315     88 :Begin {130}
316
317
318
319
320       For x:= 13 To 28 Do
321         ADW_A[ x]:= Ar429word[ x - 13 + 14];
322       ADW_A_pres[ 1]:= True;
323     End;
324     89 :Begin {131}
325       If Not ADW_A_pres[1]
326       Then Exit;
327
328       For x:= 29 To 44 Do
329         ADW_A[ x]:= Ar429word[ x - 29 + 14];
330       ADW_A_pres[ 2]:= True;

```

{ take relevant bits from  
Arinc 429 words and put  
them in an ADW. update an  
array of booleans to indi-  
cate if the complete ADW  
is present}

{ second part only valid}  
{ if first part presend}



```

331      End;
332      90 :Begin      {132}
333          If Not ADW_A_pres[2]                { third part only valid, if}
334          Then Exit;                          { second and first part}
335                                              { present}
336          For x:= 45 To 60 Do
337              ADW_A[ x]:= Ar429word[ x - 45 + 14];
338          ADW_A_pres[ 3]:= True;
339      End;
340      91 :Begin      {133}
341          If Not ADW_A_pres[3]                { fourth part only valid, if}
342          Then Exit;                          { third, second and first }
343                                              { part present}
344          For x:= 61 To 76 Do
345              ADW_A[ x]:= Ar429word[ x - 61 + 14];
346          ADW_A_pres[ 4]:= True;
347      End;
348      End;
349      If (ADW_present( ADW_A_pres) And Not Hamming_Fail( ADW_A) And
350          Not Adress_Fail( ADW_A))           { If the ADW is valid and
351                                              the address is correct,
352                                              begin decoding the info}
353      Then Begin
354          address:= ADW_Address( ADW_A);
355          Case address Of
356          1:Begin
357              Date_and_Time( Mlsint.AuxA1_time);
358              With Mlsint.auxa1 Do
359              Begin
360                  If ADW_A[ 30] = 1           { MSB is sign bit}
361                  { see Annex 10 p150}
362                  Then AzOff:= -1 * Conv_ADW( ADW_A, 21, 9)
363                  Else AzOff:= Conv_ADW( ADW_A, 21, 9);
364
365                  Az2MLSdatDist:= Conv_ADW( ADW_A, 31, 13);
366
367                  If ADW_A[55] = 1
368                  Then AzAlignRun:= -1 * Conv_ADW( ADW_A, 44, 11)
369                  Else AzAlignRun:= Conv_ADW( ADW_A, 44, 11);
370                  AzAlignRun:= AzAlignRun * 0.01;
371
372                  AzCoorSyst:= ADW_A[ 56];
373              End;
374              Mlsint.Auxa1_flag:= False;
375          End;
376          2:Begin
377              Date_and_Time( Mlsint.AuxA2_time);
378              With Mlsint.auxa2 Do
379              Begin
380                  If ADW_A[ 30] = 1
381                  Then ELOff:= -1 * Conv_ADW( ADW_A, 21, 9)
382                  Else ELOff:= Conv_ADW( ADW_A, 21, 9);
383
384                  MLSdat2thres:= Conv_ADW( ADW_A, 31, 10);
385

```

```

386      If ADW_A[ 47] = 1
387      Then ElHeight:= -1 * Conv_ADW( ADW_A, 41, 6)
388      Else ElHeight:= Conv_ADW( ADW_A, 41, 6);
389      ElHeight:= ElHeight * 0.1;
390      End;
391      Mlsint.Auxa2_flag:= False;
392      End;
393      3:Begin
394      Date_and_Time( Mlsint.AuxA3_time);
395      With Mlsint.auxa3 Do
396      Begin
397      If ADW_A[ 30] = 1
398      Then DMEoff:= -1 * Conv_ADW( ADW_A, 21, 9)
399      Else DMEoff:= Conv_ADW( ADW_A, 21, 9);
400
401      If ADW_A[ 44] = 1
402      Then DME2MLSdatdist:= -1 * Conv_ADW( ADW_A, 31, 13)
403      Else DME2MLSdatdist:= Conv_ADW( ADW_A, 31, 13);
404      End;
405      Mlsint.Auxa3_flag:= False;
406      End;
407      4:Begin
408      Date_and_Time( Mlsint.AuxA4_time);
409      With Mlsint.auxa4 Do
410      Begin
411      If ADW_A[ 30] = 1
412      Then BAZoff:= -1 * Conv_ADW( ADW_A, 21, 9)
413      Else BAZoff:= Conv_ADW( ADW_A, 21, 9);
414
415      BAZ2MLSdatdist:= Conv_ADW( ADW_A, 31, 11);
416
417      If ADW_A[ 53] = 1
418      Then BAZAlignRun:= -1 * Conv_ADW( ADW_A, 42, 11)
419      Else BAZAlignRun:= Conv_ADW( ADW_A, 42, 11);
420      BAZAlignRun:= BAZAlignRun * 0.01;
421      End;
422      Mlsint.Auxa3_flag:= False;
423      End;
424      End; { End of Case}
425
426      For x:= 1 To 4 Do
427      ADW_A_pres[x]:= False;
428      End; { End of If}
429      End; { End of procedure}
430
431
432 {-----}
433 { The following part should not necessarily be implemented, because ADW B
434   and C are not yet assigned.}
435 {-----}
436
437 Procedure ADW_B_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype;
438 a_label: byte; Var ADW_B_pres: ADW_prestype;
439 Var ADW_B: ADWtype);
440 {*****}

```

```

441 { This procedure converts the B-ADW's to Pascal variables.
442     Input : Arinc 429 word
443           Arinc 429 label
444           ADW_B_present array of boolean
445     Output: Mlsint
446           B-ADW }
447 {*****}
448 Var
449     adress      :      Byte;
450 Begin
451     Case a_label Of
452     92 :Begin      {134}
453         For x:= 13 To 28 Do
454             ADW_B[ x]:= Ar429word[ x - 13 + 14];
455             ADW_B_pres[ 1]:= True;
456         End;
457     93 :Begin      {135}
458         If Not ADW_B_pres[1]
459             Then Exit;
460
461         For x:= 29 To 44 Do
462             ADW_B[ x]:= Ar429word[ x - 29 + 14];
463             ADW_B_pres[ 2]:= True;
464         End;
465     94 :Begin      {136}
466         If Not ADW_B_pres[2]
467             Then Exit;
468
469         For x:= 45 To 60 Do
470             ADW_B[ x]:= Ar429word[ x - 45 + 14];
471             ADW_B_pres[ 3]:= True;
472         End;
473     95 :Begin      {137}
474         If Not ADW_B_pres[3]
475             Then Exit;
476
477         For x:= 61 To 76 Do
478             ADW_B[ x]:= Ar429word[ x - 61 + 14];
479             ADW_B_pres[ 4]:= True;
480         End;
481     End;
482     If (ADW_present( ADW_B_pres) And Not Hamming_Fail( ADW_B) And
483         Not Adress_Fail( ADW_B))
484     Then Begin
485         adress:= ADW_Address( ADW_B);
486         Date_and_Time( Mlsint.AuxB_time);
487         Mlsint.AuxB:= ADW_B;
488
489         For x:= 1 To 4 Do
490             ADW_B_pres[x]:= False;
491     End;
492 End;
493
494
495 Procedure ADW_C_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype;

```

```

496     a_label: byte; Var ADW_C_pres: ADW_prestype;
497     Var ADW_C: ADWtype);
498 {*****}
499 { This procedure converts the C-ADW's to Pascal variables.
500     Input : Arinc 429 word
501           Arinc 429 label
502           ADW_C_present array of boolean
503     Output: Mlsint
504           C-ADW }
505 {*****}
506 Var
507     adress      :      Byte;
508
509 Begin
510     Case a_label Of
511     96 :Begin      {140}
512         For x:= 13 To 28 Do
513             ADW_C[ x]:= Ar429word[ x - 13 + 14];
514             ADW_C_pres[ 1]:= True;
515         End;
516     97 :Begin      {141}
517         If Not ADW_C_pres[1]
518             Then Exit;
519
520         For x:= 29 To 44 Do
521             ADW_C[ x]:= Ar429word[ x - 29 + 14];
522             ADW_C_pres[ 2]:= True;
523         End;
524     98 :Begin      {142}
525         If Not ADW_C_pres[2]
526             Then Exit;
527
528         For x:= 45 To 60 Do
529             ADW_C[ x]:= Ar429word[ x - 45 + 14];
530             ADW_C_pres[ 3]:= True;
531         End;
532     99 :Begin      {143}
533         If Not ADW_C_pres[3]
534             Then Exit;
535
536         For x:= 61 To 76 Do
537             ADW_C[ x]:= Ar429word[ x - 61 + 14];
538             ADW_C_pres[ 4]:= True;
539         End;
540     End;
541     If (ADW_present( ADW_C_pres) And Not Hamming_Fail( ADW_C) And
542         Not Adress_Fail( ADW_C))      { If the ADW is valid and
543                                         the address is correct,
544                                         begin decoding the info}
545     Then Begin
546         adress:= ADW_Address( ADW_C);
547         Date_and_Time( Mlsint.AuxC_time);
548         Mlsint.AuxC:= ADW_C;
549
550         For x:= 1 To 4 Do

```

```

551         ADW_C_pres[x]:= False;
552     End;
553 End;
554 {----end of part-----}
555
556 Function Conv_BAS( ar429word: ar429wordtype; start: Byte; number: Byte)
557     : Word;
558 {*****}
559 { This function converts single bits to a number. 'Start' indicates
560   the start bit in the Arinc 429 word. 'Number' indicates the number of
561   bits to be used for the number to be formed. The number is output as a
562   Word. See Annex 10 p 60CC: LSB first.
563       Input : Arinc429 word
564             startbit
565             number of bits
566       Output: number}
567 {*****}
568 Var
569     value,
570     mult      :      Word;
571     x          :      Byte;
572
573 Begin
574     value:= 0;
575     mult:= 1;
576     For x:= 0 To number - 1 Do
577     Begin
578         value:= value + ar429word[ start + x] * mult;
579         mult:= mult * 2;
580     End;
581     Conv_BAS:= value;
582 End;
583
584
585 Procedure Bas_1_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
586 {*****}
587 { This procedure converts the Basic dataword nr 1 to Pascal variables.
588   See Annex 10 p 146.
589       Input : Arinc 429 word
590             Output: Mlsint}
591 {*****}
592 Begin
593     Date_and_Time( Mlsint.Bas1_time);
594
595     With Mlsint.Bas1 Do
596     Begin
597         Az2Thresdist:= Conv_bas( Ar429word, 12, 6) * 100;
598         AzPropCovNegLim:= Conv_bas( Ar429word, 19, 5) * 2;
599         AzPropCovPosLim:= Conv_bas( Ar429word, 24, 5) * 2;
600         Cleartype:= Ar429word[ 29];
601     End;
602     Mlsint. Bas1_flag:= False;
603 End;
604
605

```

```

606
607 Procedure Bas_2_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
608 {*****}
609 { This procedure converts the Basic dataword nr 2 to Pascal variables.
610   See Annex 10 p 146.
611       Input : Arinc 429 word
612             Output: Mlsint}
613 {*****}
614 Begin
615     Date_and_Time( Mlsint.Bas2_time);
616
617     With Mlsint.Bas2 Do
618     Begin
619         MingP:= Conv_bas( Ar429word, 13, 7) * 0.1;
620         BAZstat:= Ar429word[ 20];
621         DMEstat:= Conv_bas( ar429word, 21, 2);
622         Azstat:= Ar429word[ 23];
623         Elstat:= Ar429word[ 24];
624     End;
625     Mlsint. Bas2_flag:= False;
626 End;
627
628
629 Procedure Bas_3_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
630 {*****}
631 { This procedure converts the Basic dataword nr 3 to Pascal variables.
632   See Annex 10 p 147.
633       Input : Arinc 429 word
634             Output: Mlsint}
635 {*****}
636 Begin
637     Date_and_Time( Mlsint.Bas3_time);
638
639     With Mlsint.Bas3 Do
640     Begin
641         AzBW:= Conv_bas( Ar429word, 13, 3) * 0.5;
642         ElBW:= Conv_bas( Ar429word, 16, 3) * 0.5;
643         DMEdist:= Conv_bas( Ar429word, 19, 9) * 12.5;
644     End;
645     Mlsint. Bas3_flag:= False;
646 End;
647
648
649 Procedure Bas_4_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
650 {*****}
651 { This procedure converts the Basic dataword nr 4 to Pascal variables.
652   See Annex 10 p 147.
653       Input : Arinc 429 word
654             Output: Mlsint}
655 {*****}
656 Begin
657     Date_and_Time( Mlsint.Bas4_time);
658
659     With Mlsint.Bas4 Do
660     Begin

```

```

661      AzMagOr:= Conv_bas( Ar429word, 13, 9);
662      BazMagor:= Conv_bas( Ar429word, 22, 9);
663      End;
664      Mlsint. Bas4_flag:= False;
665      End;
666
667
668      Procedure Bas_5_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
669      {*****}
670      { This procedure converts the Basic dataword nr 5 to Pascal variables.
671      See Annex 10 p 147.
672      Input : Arinc 429 word
673      Output: Mlsint}
674      {*****}
675      Begin
676      Date_and_Time( Mlsint.Bas5_time);
677
678      With Mlsint.Bas5 Do
679      Begin
680      BAZPropCovNegLim:= Conv_bas( Ar429word, 13, 5) * 2;
681      BazPropCovPosLim:= Conv_bas( Ar429word, 18, 5) * 2;
682      BazBW:= Conv_bas( Ar429word, 23, 3) * 0.5;
683      Bazstat:= Ar429word[ 26];
684      End;
685      Mlsint. Bas5_flag:= False;
686      End;
687
688
689      Procedure Bas_6_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
690      {*****}
691      { This procedure converts the Basic dataword nr 6 to Pascal variables.
692      See Annex 10 p 147.
693      Input : Arinc 429 word
694      Output: Mlsint}
695      {*****}
696      Begin
697      Date_and_Time( Mlsint.Bas6_time);
698
699      With Mlsint.Bas6 Do
700      Begin
701      MLSident[1]:= Char( Conv_bas( ar429word, 13, 6));
702      MLSident[2]:= Char( Conv_bas( ar429word, 19, 6));
703      MLSident[3]:= Char( Conv_bas( ar429word, 25, 6));
704      End;
705      Mlsint. Bas6_flag:= False;
706      End;
707
708
709      Procedure EL_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
710      {*****}
711      { This procedure converts the Arinc 429 word containing the glidepath
712      information to a Pascal variable.
713      Input : Arinc 429 word
714      Output: Mlsint}
715      {*****}
716      Begin
717      With Mlsint Do
718      Begin
719      If ( Ar429word[ 27] = 1) And
720      ( Ar429word[ 28] = 1) And
721      ( Ar429word[ 29] = 1)
722      Then Begin
723      EAngle:= Conv_bas( Ar429word, 13, 14);
724      EAngle:= EAngle - $4000;
725      EAngle:= EAngle * 0.005;
726      End;
727
728      If ( Ar429word[ 27] = 0) And
729      ( Ar429word[ 28] = 0) And
730      ( Ar429word[ 29] = 0)
731      Then EAngle:= Conv_bas( Ar429word, 13, 14) * 0.005;
732
733      ELAntInUse:= Ar429word[ 12] + 1;      { bendix: 0 = aft ant}
734      { 1 = forward ant}
735      { ARinc: 1,2,3 = ant no}
736      EAngle_flag:= False;
737
738      End;
739      End;
740
741
742      Procedure AZ_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
743      {*****}
744      { This procedure converts the Arinc 429 word containing the azimuthangle
745      information to a Pascal variable.
746      Input : Arinc 429 word
747      Output: Mlsint}
748      {*****}
749      Var
750      x,
751      cnt      :      Byte;
752
753      Begin
754      With Mlsint Do
755      Begin
756      If ( Ar429word[ 27] = 1) And
757      ( Ar429word[ 28] = 1) And
758      ( Ar429word[ 29] = 1)
759      Then Begin
760      Azangle:= Conv_bas( Ar429word, 13, 14);
761      Azangle:= Azangle - $4000;
762      Azangle:= Azangle * 0.005;
763      End;
764
765      If ( Ar429word[ 27] = 0) And
766      ( Ar429word[ 28] = 0) And
767      ( Ar429word[ 29] = 0)
768      Then Azangle:= Conv_bas( Ar429word, 13, 14) * 0.005;
769
770      AzAntInUse:= Ar429word[ 12] + 1;      { bendix: 0 = aft ant}

```

```

771      {      1 = forward ant}
772      { ARinc: 1,2,3 = ant no}
773
774      cnt:= 0;
775      For x:= 13 to 28 Do
776          cnt:= cnt + Ar429word[ x];
777
778      If ( cnt = 0) And ( AR429word[ 29] = 1)
779      Then Leftclr:= True
780      Else Leftclr:= False;
781
782      If ( cnt = 16) And ( Ar429word[ 29] = 0)
783      Then Rightclr:= True
784      Else Rightclr:= False;
785
786      If Leftclr Or Rightclr
787      Then Azangle_flag:= True
788      Else Begin
789          AZangle_flag:= False;
790
791      End;
792  End;
793 End;
794
795 Procedure BAZ_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype);
796 {*****}
797 { This procedure converts the Arinc 429 word containing the backazimuth
798   angle information to a Pascal variable.
799   Input : Arinc 429 word
800   Output: Mlsint}
801 {*****}
802 Var
803     cnt,
804     x      :      Byte;
805
806 Begin
807     With Mlsint Do
808     Begin
809         If ( Ar429word[ 27] = 1) And
810             ( Ar429word[ 28] = 1) And
811             ( Ar429word[ 29] = 1)
812         Then Begin
813             BAZangle:= Conv_bas( Ar429word, 13, 14);
814             BAZangle:= BAZangle - $4000;
815             BAZangle:= BAZangle * 0.005;
816         End;
817
818         If ( Ar429word[ 27] = 0) And
819             ( Ar429word[ 28] = 0) And
820             ( Ar429word[ 29] = 0)
821         Then BAZangle:= Conv_bas( Ar429word, 13, 14) * 0.005;
822
823         AzAntInUse:= Ar429word[ 12] + 1;
824
825         { bendix: 0 = aft ant}
826         {      1 = forward ant}
827
828         { ARinc: 1,2,3 = ant no}
829
830         cnt:= 0;
831         For x:= 13 to 28 Do
832             cnt:= cnt + Ar429word[ x];
833
834         If ( cnt = 0) And ( AR429word[ 29] = 1)
835         Then Leftclr:= True
836         Else Leftclr:= False;
837
838         If ( cnt = 16) And ( Ar429word[ 29] = 0)
839         Then Rightclr:= True
840         Else Rightclr:= False;
841
842         If Leftclr Or Rightclr
843         Then BAZangle_flag:= True
844         Else Begin
845             BAZangle_flag:= False;
846         End;
847     End;
848 End;
849
850 Procedure Discretes_conv( Var Mlsint: Mlsinttype;
851                           Ar429word: Ar429wordtype);
852 {*****}
853 { This procedure converts the Arinc 429 word containing the MLS discretes
854   to Pascal variables.
855   Input : Arinc 429 word
856   Output: Mlsint}
857 {*****}
858 Begin
859     With Mlsint.discretes Do
860     Begin
861         antenna:= Conv_bas( Ar429word, 11, 2);
862         test      := Ar429word[ 13];
863         Azsource   := Ar429word[ 14];
864         Azselwarn  := Ar429word[ 15];
865         Bazselwarn := Ar429word[ 16];
866         GPselwarn  := Ar429word[ 17];
867         BAZavail   := Ar429word[ 18];
868         BAZdeven   := Ar429word[ 19];
869         tuningcom  := Ar429word[ 20];
870         nr1antssel := Ar429word[ 21];
871         changeinh  := Ar429word[ 22];
872         tunprtssel := Ar429word[ 23];
873     End;
874     Mlsint.discretes_flag:=False;
875 End;
876
877 Function Hex2String( x: byte): string;
878 {*****}
879 { This function converts a byte to its hexadecimal representation in a
880   string. Written by Rob Luxen.
881   Input : Byte;

```

```

881                               Output: A string;
882 {*****}
883 Const
884     hex: Array[ 0..15] Of Char = '0123456789ABCDEF';
885
886 Var
887     i :      Byte;
888
889 Begin
890     Hex2String[ 0]:= Chr(2);
891     For i:= 0 to 1 Do
892     Begin
893         hex2String[2-I]:= Hex[ x and $000F];
894         x:= x shr 4;
895     End;
896 End;
897 {-----end included procedures CollectMLSrec-----}
898
899 Var
900     line,
901     part      :   String;
902     Ar429word :   Ar429wordtype;
903     a_label   :   Byte;
904     adress    :   Byte;
905     NoWord    :   Boolean;
906     time,
907     result    :   timetype;
908     d,d1      :   Byte;
909
910 {-----Start procedure CollectMLSrec-----}
911 Begin      { begin procedure CollectMLSrec}
912     MLSint:= tempMLSint;
913
914     Ar429comm.GetAr429word( Ar429word, NoWord, a_label);
915                               { the label is in dec}
916 Gotoxy( 1, 13);
917 If Not ( Noword)
918 Then Begin
919     Str( a_label: 3, part);
920     line:= part + ' ';
921     For x:= 1 To 32 Do
922     Begin
923         Str( ar429word[x]:1, part);
924         line:= line + part;
925     End;
926     Write( line, ' ');
927
928     SaveEquipmentMessage( 'RM: ' + line);
929 End;
930
931     Ar429.GetAr429word( Ar429word, NoWord, a_label);
932                               { the label is in dec}
933 If Not ( Noword) Then Begin
934     Str( a_label: 3, part);
935     line:= part + ' ';

```

```

936 For x:= 1 To 32 Do
937 Begin
938     Str( ar429word[x]:1, part);
939     line:= line + part;
940 End;
941 WriteLn( line);
942
943 SaveEquipmentMessage( 'RM: ' + line);
944 End;
945
946 With Fifo Do
947 Begin
948     If Not FifoEmpty
949     Then Begin
950         d:= GetFifo;
951         If d = $AA
952         Then Begin
953             If Not FifoEmpty
954             Then Begin
955                 d1:= GetFifo;
956                 If d1 = $55
957                 Then Begin
958                     SaveEquipmentMessage( 'RM: ' + badwline);
959                     badwline:= '';
960                     End
961                     Else Begin
962                         badwline:= badwline +
963                             Hex2String( d) +
964                             ' ';
965                         badwline:= badwline +
966                             Hex2String( d1) +
967                             ' ';
968                     End;
969                     End;
970                     Else Begin
971                         badwline:= badwline +
972                             Hex2String( d) +
973                             ' ';
974                     End;
975                     End;
976                     End;
977                     tempMLSint:= MLSint;
978 End;
979 End;
980 {-----End procedure CollectMLSrec-----}
981
982
983
984
985 {-----Start procedure Execmlsrecommand-----}
986 Procedure ExecMLSrecCommand( command: commandtype);
987
988 Const
989     Prate      =      1000;
990

```

```

991 Var
992   error      :      Boolean;
993   datain     :      Longint;
994   oct_lab    :      Byte;
995   part,
996   line       :      String;
997   ar429word  :      ar429wordtype;
998
999 Begin
1000   { room for decoding the command. The result should be a valid
1001     arinc429 word of 32 bits. This should be coded in a 32 bit longint}
1002
1003   Ar429.SendAr429word( datain, oct_lab, Prate, error);
1004
1005   line:= '';
1006   For x:= 1 To 32 Do
1007   Begin
1008     Str( ar429word[x]:1, part);
1009     line:= line + part;
1010   End;
1011   SaveEquipmentMessage( 'SM: ' + line);
1012 End;
1013 {-----End procedure Execmlsrecommand-----}
1014
1015
1016 {-----Start procedure Closemlsrec-----}
1017 Procedure CloseMLSrec;
1018
1019 Var
1020   setupfile  :      Text;
1021   value      :      String;
1022
1023 Begin
1024   Ar429comm.CloseAR429;
1025   Ar429.CloseAr429;
1026   Remove_ADW_int;
1027
1028   OpenConfigWrite( setupfile, MIAScfname);
1029   Writeln( setupfile, 'MLSBENDIX');
1030
1031   Str( completeinfo, value);
1032   Writeln( setupfile, #9'completeinfo = ', value, ';');
1033
1034   Str( cardaddress, value);
1035   Writeln( setupfile, #9'cardaddress = ', value, ';');
1036
1037   Str( irq, value);
1038   Writeln( setupfile, #9'irq = ', value, ';');
1039
1040   CloseConfig( setupfile);
1041 End;
1042 {-----End procedure Closemlsrec-----}
1043
1044
1045 {-----Start initialising-----}

```

```

1046 Begin
1047   For x:= 1 To 4 Do
1048   Begin
1049     ADW_A_pres[x]:= False;
1050     ADW_B_pres[x]:= False;
1051     ADW_C_pres[x]:= False;
1052   End;
1053   For x:= 13 To 76 Do
1054   Begin
1055     ADW_A[ x]:= 0;
1056     ADW_B[ x]:= 0;
1057     ADW_C[ x]:= 0;
1058   End;
1059
1060   With TempMLSint Do
1061   Begin
1062     Bas1_flag    := True;
1063     Bas2_flag    := true;
1064     Bas3_flag    := true;
1065     Bas4_flag    := true;
1066     Bas5_flag    := true;
1067     Bas6_flag    := true;
1068     auxa1_flag   := true;
1069     auxa2_flag   := true;
1070     auxa3_flag   := true;
1071     auxa4_flag   := true;
1072     ELangle_flag := true;
1073     AZangle_flag := true;
1074     BAZangle_flag:= true;
1075     DME_flag     := true;
1076     discretes_flag:= true;
1077     leftclr      := true;
1078     rightclr     := true;
1079     flag         := true;
1080
1081     ErrorTime( Bas1_time);
1082     ErrorTime( Bas2_time);
1083     ErrorTime( Bas3_time);
1084     ErrorTime( Bas4_time);
1085     ErrorTime( Bas5_time);
1086     ErrorTime( Bas6_time);
1087     ErrorTime( AuxA1_time);
1088     ErrorTime( AuxA2_time);
1089     ErrorTime( AuxA3_time);
1090     ErrorTime( AuxA4_time);
1091     ErrorTime( AuxB_time);
1092     ErrorTime( AuxC_time);
1093   End;
1094
1095   ErrorTime( Tangle);
1096   ErrorTime( Valid_Tangle);
1097   Valid_Tangle.sec100:= 8;
1098
1099   anglebegin:= False;
1100   completeinfo:= 1;

```

{ angles are no longer}  
{ valid than 80 msec}

Page 11, listing of MLSBENDI.PAS, date is 18-02-93, file date is 17-02-93, size is 41146 bytes.

1101 End.

1102 {-----End initialising-----}

1103 {-----End Unit MLSBENDIX-----}



```

1 Unit AttBeaver;
2
3 {$N+,E+}
4
5 Interface
6
7 Uses MIASglob;
8
9 Procedure InitAttTX( Var error: Boolean);
10 {*****}
11 { This procedure initialises the ATTitude transmitter}
12 {
13     Input : -
14     Output: -}
15 {*****}
16 Procedure CollectAtt( Var attdata: attdatatype);
17 {*****}
18 { This procedure collects the attitude angles}
19 {
20     Input : -
21     Output: attitude angles in radians}
22 {*****}
23 Procedure ExecAttTxcommand( command: commandtype);
24 {*****}
25 { This procedure passes a command to the Attitude transmitter.
26   NotE: This procedure is empty}
27 {
28     Input : -
29     Output: -}
30 {*****}
31 Procedure CloseAttTx;
32 {*****}
33 { This procedure restores the ATTitude transmitter to its original state}
34 {
35     Input : -
36     Output: -}
37 {*****}
38
39 Implementation
40
41
42 Uses Miscell, Synchcnv, crt, user;
43
44
45 Const
46   pi      =      3.1415926535897932385;
47
48
49 Var
50   PitchOffset ,
51   RollOffset  :      Double;
52
53 {-----start procedure initatttx -----}
54 Procedure InitAttTX( Var error: Boolean);
55

```

```

56 Var
57   setupfile   :      Text;
58   title,
59   varname,
60   line        :      String;
61   value       :      String;
62   code        :      Integer;
63
64 Begin
65   OpenConfigRead( setupfile, MIAScfname);
66   Repeat
67   { find attbeaver part of
68     config file}
69     Readln( setupfile, title);
70   Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'ATTBEAVER'));
71
72   If Not Eof( setupfile)
73   { if there is more in file}
74   Then Repeat
75     Readln( setupfile, line);
76     Convert( line, varname, value); { extract the variable name
77                                     and value}
78     If ( varname = 'PITCHOFFSET')
79     Then Val( value, pitchoffset, code);
80     If ( varname = 'ROLLOFFSET')
81     Then Val( value, rolloffset, code);
82     Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
83   CloseConfig( setupfile);
84
85   InitSynchcnv( error);
86 End;
87 {-----end procedure initatttx -----}
88
89
90 {-----Start procedure collectatt -----}
91 Procedure CollectAtt( Var attdata: attdatatype);
92
93 Var
94   error       :      Boolean;
95   part,
96   line        :      String;
97   time        :      Timetype;
98
99 Begin
100   attdata.flag:= True;
101
102   Roll_Pitch_heading( 1, attdata.rollangle, error);
103   attdata.flag:= error;
104   Roll_Pitch_heading( 2, attdata.pitchangle, error);
105   attdata.flag:= attdata.flag Or error;
106
107   If Not ( error) Then Begin
108     Gotoxy( 1, 11);
109     Str( attdata.rollangle: 6: 2, part);
110     Write( 'rollangle = ', part);

```

```
111 line:= part + ' ';
112
113 Str( attdata.pitchangle: 6: 2, part);
114 Write( ' pitchangle = ', part);
115 line:= line + part + ' ';
116
117 SaveEquipmentMessage( 'RA: ' + line);
118 End;
119
120 attdata.rollangle:= ( attdata.rollangle - rolloffset) * pi / 180;
121 attdata.pitchangle:= ( attdata.pitchangle - pitchoffset) * pi / 180;
122 End;
123 {-----end procedure collectatt -----}
124
125
126 {-----start procedure execatttxcommand-----}
127 Procedure ExecAttTxcommand( command: commandtype);
128
129 Begin
130 End;
131 {-----end procedure execatttxcommand-----}
132
133
134 {-----start procedure closeatttx-----}
135 Procedure CloseAttTx;
136
137 Var
138     setupfile      :      Text;
139     Value           :      String;
140
141 Begin
142     CloseSynchcnv;
143
144     OpenConfigWrite( setupfile, MIAScfname);
145     Writeln( setupfile, 'ATTBEAVER');
146
147     Str( Pitchoffset, value);
148     Writeln( setupfile, #9'pitchoffset = ', value, ';');
149
150     Str( Rolloffset, value);
151     Writeln( setupfile, #9'rolloffset = ', value, ';');
152
153     CloseConfig( setupfile);
154 End;
155 {-----End procedure closeatttx-----}
156
157 {-----Start initialising-----}
158 Begin
159     PitchOffset:= 0;
160     RollOffset:= 0;
161 End.
162 {-----End initialising-----}
163 {-----End Unit ATTBEAVER-----}
```

```

1 Unit hdgBeaver;
2
3 {$N+,E+}
4
5 Interface
6
7 Uses MIASglob;
8
9 Procedure InitHdgTX( Var error: Boolean);
10 {*****}
11 { This procedure initialises the hdgitude transmitter}
12 {*****}
13
14 Procedure CollectHdg( Var hdgdata: hdgdatatype);
15 {*****}
16 { This procedure collects the hdgitude data}
17 {*****}
18
19 Procedure ExechdgTxcommand( command: commandtype);
20 {*****}
21 { This procedure passes a command to the hdgitude transmitter.
22   NotE: This procedure is empty}
23 {*****}
24
25 Procedure CloseHdgTx;
26 {*****}
27 { This procedure restores the hdgitude transmitter to its original state}
28 {*****}
29
30
31 Implementation
32
33
34 Uses Miscell, Synchcnv, crt, user;
35
36 Const
37     pi      =      3.1415926535897932385;
38
39 Var
40     HdgOffset :      Double;
41
42 {----- start procedure inithdgtx -----}
43 Procedure InitHdgTX( Var error: Boolean);
44
45 Var
46     setupfile :      Text;
47     title,
48     varname,
49     line      :      String;
50     value     :      String;
51     code      :      Integer;
52
53 Begin
54     OpenConfigRead( setupfile, MIAScfname);
55     Repeat          { find headingbeaver part of

```

```

56                                     config file}
57     Readln( setupfile, title);
58     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'HDGBEAVER'));
59
60     If Not Eof( setupfile)          { if there is more in file}
61     Then Repeat
62         Readln( setupfile, line);    { get a line}
63         Convert( line, varname, value); { extract the variable name
64                                         and value}
65
66         If ( varname = 'HDGOFFSET')
67         Then Val( value, hdgoffset, code);
68         Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
69     CloseConfig( setupfile);
70
71     InitSynchcnv( error);
72 End;
73 {----- end procedure inithdgtx -----}
74
75
76 {----- start procedure collecthdg -----}
77 Procedure CollectHdg( Var hdgdata: hdgdatatype);
78
79 Var
80     error      :      Boolean;
81     part,
82     line       :      String;
83     time       :      Timetype;
84
85 Begin
86     hdgdata.flag:= True;
87
88     Roll_pitch_heading( 3, hdgdata.hdgangle, error);
89
90     hdgdata.flag:= error;
91
92 If Not ( error) Then Begin
93     Gotoxy( 40, 11);
94     Str( hdgdata.hdgangle: 6: 2, part);
95     Write( 'hdgangle = ', part);
96     line:= part + ' ';
97
98     SaveEquipmentMessage( 'RH: ' + line);
99 End;
100     hdgdata.hdgangle:= ( hdgdata.hdgangle - hdgoffset) * pi / 180;
101 End;
102 {----- End procedure collecthdg-----}
103
104
105 {----- start procedure exechdgtxcommand-----}
106 Procedure ExechdgTxcommand( command: commandtype);
107
108 Begin
109 End;
110 {----- End procedure exechdgtxcommand-----}

```

```
111
112
113 {----- Start procedure Closehdgtx-----}
114 Procedure ClosehdgTx;
115
116 Var
117     setupfile      :      Text;
118     Value           :      String;
119
120 Begin
121     CloseSynchcnv;
122
123     OpenConfigWrite( setupfile, MIAScfname);
124     Writeln( setupfile, 'HDGBEAVER');
125
126     Str( Hdgoffset, value);
127     Writeln( setupfile, #9'hdgoffset = ', value, ';');
128
129     CloseConfig( setupfile);
130
131 End;
132 {----- End procedure Closehdgtx-----}
133
134
135 {----- start initialising-----}
136 Begin
137     hdgoffset:= 0;
138 End.
139 {----- End initialising-----}
140 {----- End unit HDGBEAVE-----}
```

```

1 Unit Key_Cons;
2
3 Interface
4
5 Uses MIASglob;
6
7 Procedure OpenIn_OutputDev( MIASlogname: String);
8 {*****}
9 { opens a file for output.}
10 {
11     Input : -
12     Output: -}
13 {*****}
14 Procedure GetMessage( Var command: commandtype);
15 {*****}
16 { This procedure retrieves a message from the keyboard. The procedure will
17   return a string as soon as the 'ENTER' key is pressed.
18   Input : keystrokes
19   Output: string with keystrokes}
20 {*****}
21
22 Procedure SendMessage( command: commandtype);
23 {*****}
24 { This procedure sends a message to the display connected to the computer.
25   This is the standard display.
26   Input : string with message
27   Output: string with message on screen.}
28 {*****}
29
30 Procedure SendFlags( command: commandtype);
31 {*****}
32 { This procedure sends a message ( flags) to a prescribed position on
33   the screen.
34   Input : flags
35   Output: flags on the screen}
36 {*****}
37
38
39 Procedure SaveMessage( message: commandtype);
40 {*****}
41 { this procedure saves a message to disc.
42   Input : message
43   Output: message on disc}
44 {*****}
45
46
47 Procedure CloseIn_OutputDev;
48 {*****}
49 { closes the file for output.}
50 {
51     Input : -
52     Output: -}
53 {*****}
54 Implementation
55

```

```

56 Uses Dos, Crt, Miscell;
57
58 Const
59     Cr      =    #13;
60
61 Var
62     inputmessage :    commandtype;
63     outputfile   :    Text;
64     outname      :    String;
65     Disknr       :    Byte;
66     Free         :    Longint;
67     blockused    :    Longint;
68
69
70 Procedure SendMessageToDisplay( command: commandtype);
71
72 Begin
73     Gotoxy( 1, 21);
74     Write( command);
75     If ( command[ length( command)] = #10) Or  { LF}
76       ( command[ length( command)] = #13)    { CR}
77     Then Gotoxy( Length( command), 21)
78     Else Gotoxy( Length( command) + 1, 21);
79     Write( '
80 End;
81
82
83 Procedure OpenIn_OutputDev( MIASlogname: String);
84
85 Begin                                     { init for Key_cons.int}
86     ClrScr;
87     inputmessage:= '';
88
89     If ( MIASlogname[2] = ':') And ( Length( MIASlogname) >=2)
90     Then Case MIASlogname[1] Of
91         'a', 'A': Disknr:= 1;
92         'b', 'B': Disknr:= 2;
93         'c', 'C': Disknr:= 3;
94     End
95     Else Disknr:= 0;
96
97     SendMessageToDisplay( 'Insert empty disc '+
98                           'and press Enter when ready'#10#13+
99                           'Disc should contain "MIAS.CFG"');
100
101     Readln;
102
103     outname:= MIASlogname + 'mias.dat';
104     Assign( outputfile, outname);          { defined in MIASglob}
105
106     If FileExist( outname)
107     Then Append( outputfile)
108     Else Rewrite( outputfile);
109
110     Free:= DiskFree( disknr);
111     Blockused:= 0;

```

```

111 End;
112
113
114 Procedure GetMessage( Var command: commandtype);
115
116 Var
117   ch   :   Char;
118
119 Begin
120   If Keypressed
121     Then Begin
122       gotoxy( 1, 23);
123       ch:= Ucase( readkey);
124
125       If ch = #8           { backspace}
126       Then inputmessage:= Copy( inputmessage, 1,
127                                Length( inputmessage) -1)
128       Else inputmessage:= inputmessage + ch;
129       { Enter}
130       If inputmessage[ Length( inputmessage)] = CR
131       Then Begin
132         command:= Copy( inputmessage, 1,
133                        Length( inputmessage) -1);
134         inputmessage:= '';
135         SendMessage( command);
136       End
137       Else command:= '';
138       Write( inputmessage);
139       Write( '
140     End
141   Else command:= '';
142 End;
143
144
145 Procedure SendMessage( command: commandtype);
146
147 Begin
148   SendMessageToDisplay( command);
149   SaveMessage( command);
150 End;
151
152
153 Procedure SendFlags( command: commandtype);
154
155 Begin
156   Gotoxy( 1, 19);
157   Write( command);
158 End;
159
160 Procedure SaveMessage( message: commandtype);
161
162 Var
163   hour,
164   minute,
165   sec,

```

```

166   sec100      :   Word;
167   part,
168   line        :   String;
169   seconds     :   Longint;
170
171 Begin
172   GetTime( hour, minute, sec, sec100);
173   seconds:= hour * 60;
174   seconds:= seconds + minute;
175   seconds:= seconds * 60;
176   seconds:= seconds + sec;
177   seconds:= seconds * 100;
178   seconds:= seconds + sec100;
179
180   str( seconds: 7, part);
181   line:= part + ' ';
182
183   While ( message[ length( message)] = #10) Or
184         ( message[ length( message)] = #13) Do
185     message:= Copy( message, 1, Length( message) -1);
186
187   Writeln( outputfile, line + message);
188
189   Blockused:= Blockused + Length( line+message) + 2;
190   If Blockused >= 10000
191   Then Begin
192     Flush( outputfile);
193     Close( outputfile);
194     Append( outputfile);
195     Blockused:= 0;
196   End;
197
198   Free:= Free - Length( line+message) - 2;
199   If (Free <= 512) And ( Free >= 0)
200   Then Begin
201     Flush( outputfile);
202     Close( outputfile);
203     SendMessageToDisplay( 'Disc full. Insert empty disc '+
204                          'and press Enter when ready!');
205     Readln;
206     Free:= DiskFree( disknr);
207     Assign( outputfile, outname);
208     Rewrite( outputfile);
209   End;
210 End;
211
212
213 Procedure CloseIn_OutputDev;
214
215 Begin
216   Close( outputfile);
217   { cannot be checked effect}
218   { ively with 'IOresult'}
219 End;
220 Begin

```

Page 3, listing of KEY\_CONS.PAS, date is 18-02-93, file date is 05-01-93, size is 6764 bytes.

221 End.

```

1 Unit Synchcnv;
2 {*****}
3 { This unit performs interfacing with the synchroconverter card.
4   The procedure 'ROLL_PITCH_HEADING' was written by Dennis Willemsen.
5
6   The rest of this unit was written by Marco Meijer}
7 {*****}
8
9 {$N+,E+}
10
11 Interface
12
13 Uses MIASglob;
14
15
16 Procedure InitSynchcnv( Var error: Boolean);
17 {*****}
18 { This procedure resets the synchroconverter card.}
19 {
20   Input : -
21   Output: -}
22 {*****}
23
24 PROCEDURE roll_pitch_heading (number:INTEGER;VAR angle:Double;
25   VAR error:BOOLEAN);
26 {*****}
27 { This procedure instructs the synchroconverter card to output the angle
28   specified by the number: nummber. The angle is output in the variable
29   hoek. The angle is output in degrees.
30   Input : angle number
31   output: angle}
32 {*****}
33
34
35 Procedure CloseSynchcnv;
36 {*****}
37 { This procedure restores the synchroconverter card to its original state.
38   Input : -
39   Output: -}
40 {*****}
41
42
43 Implementation
44
45
46 {*****}
47 {* This function replaces the bits, which means that bit 8 will switch *}
48 {* place with bit 1, bit 7 will switch place with bit 2, etc. This is *}
49 {* done by first determining the value of the bit and then multiply it *}
50 {* with the value of the place it should be on. The new value of the byte *}
51 {* is obtained by adding all the separate value's. *}
52 {*****}
53
54
55 FUNCTION shift (old:BYTE):BYTE;

```

```

56
57 VAR new, conversion      :   BYTE;
58     power, p, weight     :   INTEGER;
59
60 BEGIN
61   weight := 1;
62   power := 128;
63   new := 0;
64   shift := 0;
65   conversion := 0;
66   FOR p := 1 to 8 DO
67     BEGIN
68       new :=((old AND weight) * power) DIV weight;
69       conversion := conversion + new;
70       power := power DIV 2;
71       weight := weight * 2;
72     END;
73   shift := conversion;
74 END;
75
76
77 Procedure InitSynchcnv( Var error: Boolean);
78
79 Var
80   result      :   Boolean;
81   x           :   Integer;
82   hoek        :   Double;
83
84 Begin
85   result:= True;
86   For x:= 1 To 3 Do
87     Begin
88       roll_pitch_heading( x, hoek, error);
89       result:= result And error;
90     End;
91   error:= result;
92 End;
93
94
95 {*****}
96 {* This procedure gives the value's of one of three synchro's. The dif- *}
97 {* ferent synchro's are selected by addressing the procedure with the *}
98 {* numbers 1,2 or 3. It will reset the synchro's and it will read each *}
99 {* byte 2 times, so that error tests can be performed. The bytes that *}
100 {* are read first have to be shifted and this is done by the function *}
101 {* shift. The hardware card is addressed by the addresses $300-$30F. The *}
102 {* stepsize is obtained by deviding 360 by (2^16 - 1). *}
103 {*****}
104
105 PROCEDURE roll_pitch_heading (number:INTEGER;VAR angle:Double;
106   VAR error:BOOLEAN);
107
108 CONST
109   roll1    = $300;
110   roll2    = $308;

```



```

111 pitch1  = $304;
112 pitch2  = $30C;
113 heading1 = $302;
114 heading2 = $30A;
115 reset    = $30F;
116 stepsize = 5.493080245e-3;
117
118 VAR
119   b                : INTEGER;
120   amount_of_steps  : WORD;
121   lsb1, msb1, msb, lsb, rsb,
122   lsbc, msbc       : BYTE;
123
124 BEGIN
125   angle := 0;
126   amount_of_steps := 0;
127   rsb := PORT[reset];
128   CASE number OF
129     1: BEGIN
130       rsb := PORT[reset];
131       msb1 := PORT[roll1];
132       lsb1 := PORT[roll2];
133       msbc := PORT[roll1];
134       lsbc := PORT[roll2];
135     END;
136     2: BEGIN
137       rsb := PORT[reset];
138       msb1 := PORT[pitch1];
139       lsb1 := PORT[pitch2];
140       msbc := PORT[pitch1];
141       lsbc := PORT[pitch2];
142     END;
143     3: BEGIN
144       rsb := PORT[reset];
145       msb1 := PORT[heading1];
146       lsb1 := PORT[heading2];
147       msbc := PORT[heading1];
148       lsbc := PORT[heading2];
149     END;
150   END;
151
152   lsb := shift(lsb1);
153   msb := shift(msb1);
154   amount_of_steps := (lsb + msb * 256);
155   angle := amount_of_steps * stepsize;
156
157
158   {*****}
159   {* This is where the error-testing starts. First the values of the *}
160   {* bytes are tested on their correctness. Of the lsb1 only the 7 least *}
161   {* significant bits are compared, because the 8th bit alters al the *}
162   {* time, this is due to the accuracy of the chip. Also the amount of *}
163   {* steps is checked as wel as the angle. *}
164   {*****}
165
166   lsbc := lsbc AND 127;
167   lsb1 := lsb1 AND 127;
168   IF (lsb = lsbc) AND (msb = msbc)
169     THEN error := FALSE
170     ELSE error := TRUE;
171
172   IF ((amount_of_steps < 0) OR (amount_of_steps > 65535))
173     AND (error = FALSE)
174     THEN error := TRUE
175     ELSE error := FALSE;
176
177   IF ((angle < 0) OR (angle >= 360)) AND (error = FALSE)
178     THEN error := TRUE
179     ELSE error := FALSE;
180
181 END;
182
183
184
185 Procedure CloseSynchcnv;
186
187 Begin
188 End;
189
190 End.
```

```

1 Unit AR429;
2
3
4 Interface
5
6 Type
7     AR429wordtype      =      Array[1..32] Of Byte;
8     arraytype          =      Array[ 0..255] Of Byte;
9
10 Procedure InitAR429( Var error: Boolean; selecttable: arraytype; num: Integer);
11 {*****}
12 { This procedure installs the ARINC 429 handler. It catches the transmissions
13   from the external ARINC 429 channel. The label numbers to be caught should
14   be programmed in selecttable in octal. The number of labels should be
15   passed with num. Error indicates, that something went wrong during
16   initialising of the receiver.
17       Input : selecttable
18               number of labels in table
19       Output: error}
20 {*****}
21
22
23 Procedure GetAr429word( Var ar429word: AR429wordtype; Var NoWord: Boolean;
24   Var a_label: Byte);
25 {*****}
26 { This procedure delivers one ARINC 429 word. If no word is available,
27   the NoWord-flag will be set True. The label will be output also. If
28   no word is available, the word is set to all zero's and a_label = 0.
29       Input : -
30       Output: Arinc 429 word
31               NoWord flag
32               adress label}
33 {*****}
34
35
36 Procedure SendAr429word( dataIn: Longint; oct_lab: Byte; Prate: Word;
37   Var error: Boolean);
38 {*****}
39 { This procedure sends a command to an external ARINC 429 channel.
40       Input : ARINC 429 word
41       Output: ARINC 429 word on 429 channel}
42 {*****}
43
44
45 Procedure CloseAR429;
46 {*****}
47 { This procedure makes sure, the computer and peripherals are in their
48   starting state.
49       Input : -
50       Output: -}
51 {*****}
52
53
54 Implementation
55
56 Uses Lib429P;
57
58 Const
59     Pspeed      =      S100;           { 100 Kbits per sec}
60     Pgap        =      G4_00;         { gap time of 4 bittimes}
61                                           { NOTE: LABELS ARE OCTAL}
62     ssm         =      0;             { sign status matrix}
63     sdi         =      0;             { source destination indicator}
64     max429buf   =      100;
65     parity      =      Podd;
66     on          =      1;
67     off         =      0;
68
69
70 Var
71     nextfreeslot :      Byte;
72     arinc429buffer:      Array[0..Max429buf-1] Of Longint;
73     inp          :      Integer;
74     outp         :      integer;
75     arTxChannel  :      Byte;
76     ptr1,
77     ptr2         :      Byte;
78
79 Procedure InitAR429( Var error: Boolean; selecttable: arraytype; num: Integer);
80
81 Var
82     x            :      Byte;
83     Rspeed,
84     Rgap         :      Integer;
85     readtable    :      arraytype;
86
87 Begin
88     error:= False;
89
90     arSetBase($D000, 0, 0, 0);       { set card base address}
91     arReset(0);                      { Turn OFF all channels,
92                                           all blocks and all
93                                           receiver modes}
94                                           { INITIALISE TRANSMITTER}
95     arTxSetSpeed( 0, Pspeed, Pgap);   { set transmitter speed}
96     arTxReadSpeed( 0, Rspeed, Rgap);
97
98     If ( Pspeed <> Rspeed) Or ( Pgap <> Rgap) { verify transmitter speed}
99     Then Begin
100         error:= True;
101         Exit;
102     End;
103
104     arTxSetParity( 0, Pnone);         { set transmitter channel}
105                                           { parity to none, to allow}
106                                           { other data to override}
107
108     nextfreeslot:= 0;
109     arTxChannel := off;
110
111 { INITIALISE RECEIVER}

```

```

111 arRxStop( 0);                { stop receiver}                166
112 arRxSpeedHigh( 0);          { set receiver on high speed}    167
113                               168
114 For x:= 0 To num -1 Do      169
115     selecttable[ x]:= oct( selecttable[ x]); { convert labels to decimal} 170
116                               171
117 arRxSelectSet( 0, @Selecttable, num); { select the labels to be read} 172
118                               173
119 arRxSelectRead( 0, @ReadTable); { verify the labels to be read} 174
120                               175
121 For x:= 0 To num -1 Do      176
122     If readtable[ selecttable[ x]] = 0 { with respect to programmed} 177
123     Then Begin              { labels} 178
124         error:= True;      179
125         Exit;              180
126     End;                   181
127                               182
128 arRxConfigWrite( 0, circular, shared, 42, nosync, 0, 0, 0); 183
129                               { configure the receiver} 184
130 arRxStatTrig( 0);           { start the receiver} 185
131 ptr1:= arRxGetPtr( 0);      186
132                               187
133 outp:=0;                   188
134 inp:= 0;                   189
135 End;                       190
136                               191
137                               192
138 Procedure GetAr429word( Var ar429word: AR429wordtype; Var NoWord: Boolean; 193
139     Var a_label: Byte);      194
140                               195
141 Type                         196
142     stat_buffer =            197
143         Record               Var
144             data: Longint;    198
145             tagtime: word;    199
146         End;                 200
147 Var                         201
148     x : Integer;             202
149     arincw : Longint;        203
150     Realarincw : Real;       204
151     temp : Real;             205
152     time : word;             206
153                               207
154                               208
155 Begin                       209
156     ptr2:= arRxGetPtr( 0);    210
157     While (ptr1 <> ptr2) Do    211
158     Begin                    212
159         ptr1:= arRxBuffer( 0, ptr1, arinc429buffer[ inp], time); 213
160         inp:= ( inp + 1 ) Mod max429buf; 214
161     End;                     215
162                               216
163 If outp <> Inp                { if word in buffer} 217
164 Then Begin                  218
165     arincw:= arinc429buffer[ outp]; { get word} 219

```

```

166 outp:= (outp + 1 ) Mod max429buf; { increment outpointer}
167
168 a_label:= arLabel( arincw); { get label}
169
170
171 realarincw:= arincw;
172 temp:= 1; { decode longint to bits}
173 For x:= 1 to 32 do
174     temp:= temp * 2;
175
176 If arincw < 0
177 Then realarincw:= realarincw + temp;{ make positive}
178
179 temp:= temp / 2;
180 For x:= 32 DownTo 1 Do { 32 bits}
181 Begin
182     ar429word[ x]:= Trunc( realarincw / temp);
183     If ar429word[x] > 0 { test on specific bit}
184     Then realarincw:= Round( realarincw - temp);
185     temp:= Trunc (temp / 2); { select next bit}
186 End;
187 noword:= False;
188
189 End
190 Else noword:= True;
191
192 End;
193
194 Procedure SendAr429word( dataIn: Longint; oct_lab: Byte; Prate: Word;
195     Var error: Boolean);
196
197 Var
198     Rstatus : Byte;
199     Rrate : Word;
200     Rarincw : Longint;
201     Rlab : integer;
202     Rdata : Longint;
203     Rssm,
204     Rsdi,
205     Rpar : Integer;
206     Parincw : Longint;
207     Pstatus : Byte;
208     done : Boolean;
209     x : Integer;
210
211 Begin
212     arCompose( Parincw, oct( oct_lab), dataIn, ssm, sdi, parity);
213
214     If ( parity = Podd) { program the status mask}
215     Then Pstatus:= $00 { for parity}
216     Else Pstatus:= $01;
217
218     Pstatus:= Pstatus Or $80; { program status mask for}
219     { 'slot on'}
220     error:= False;

```

```

221 done:= False;           { ready inserting word?}           276 Procedure CloseAR429;
222 For x:= 0 To nextfreeslot -1 Do { check all used slots for} 277
223 Begin                                     278 Begin
224     arTxReadSlot( 0, x, Rstatus, Rrate, Rarincw);           279     arReset( 0);           { turn off channel 0}
225     arDecompose( Rarincw, Rlab, Rdata, Rssm, Rsd1, Rpar);    280     { all slots and all receiver}
226                                     281     { modes}
227     If ( oct( oct_lab) = Rlab) { labels already in use}    282 End;
228     Then Begin                                     283
229         arTxSetdata( 0, x, Parincw); { update these labels} 284
230         arTxSetRate( 0, x, Prate);
231         arTxReadSlot( 0, x, Rstatus, Rrate, Rarincw);        285 Begin
232                                     286 End.
233     done:= True;
234     If ( (Rstatus And Pstatus) <> Pstatus) Or
235         { verify transmitter}
236         ( Rrate <> Prate) Or { contents}
237         ( Rarincw <> Parincw)
238     Then Begin
239         error:= True;
240         Exit;
241     End;
242 End;
243 End;
244
245 If done = False
246 Then Begin
247     arTxSetdata( 0, nextfreeslot, Parincw);{ put data in next free slot}
248     { on the channel}
249     arTxSetRate( 0, nextfreeslot, Prate); { set the transmit rate for}
250     { slot 0}
251     arTxSlotOn( 0, nextfreeslot); { enable slot 0}
252
253     arTxReadSlot( 0, nextfreeslot, Rstatus, Rrate, Rarincw);
254
255     If ( (Rstatus And Pstatus) <> Pstatus) Or
256         { verify transmitter}
257         ( Rrate <> Prate) Or { contents}
258         ( Rarincw <> Parincw)
259     Then Begin
260         error:= True;
261         Exit;
262     End;
263
264     If ( arTxChannel = off)
265     Then Begin
266         arTxChannelOn( 0); { turn channel on}
267         arTxChannel:= On;
268     End;
269
270     nextfreeslot:= (nextfreeslot + 1) Mod 63;
271     { if 63 slots active, over-}
272     { write slot 0}
273 End;
274
275

```

```

1 Unit AR429comm;
2 {*****}
3 { This unit contains routines for decoding and interfacing with an Arinc
4   429 datastream. The datastream is input using a comport}
5 {*****}
6
7 Interface
8
9 Uses AR429; { This statement is placed here instead of: "
10              Type AR429wordtype = Array[1..32] Of Byte;"
11              This statement caused compile time errors due to
12              duplicate type declarations with the same name.
13              The first declaration is done in AR429comm.
14              If AR429comm is not used, replaced the statement}
15
16
17 Procedure InitAR429;
18 {*****}
19 { This procedure installs the ARINC 429 handler. It catches the transmissions
20   from the external ARINC 429 channel.
21   Input : -
22   Output: -}
23 {*****}
24
25
26 Procedure GetAr429word( Var ar429word: AR429wordtype; Var NoWord: Boolean;
27                        Var a_label: Byte);
28 {*****}
29 { This procedure delivers one ARINC 429 word. If no word is available,
30   the NoWord-flag will be set True. The label will be output also. If
31   no word is available, the word is set to all zero's and a_label = 0.
32   Input : -
33   Output: Arinc 429 word
34           NoWord flag
35           adress label}
36 {*****}
37
38
39 Procedure SendAr429word( word: AR429wordtype);
40 {*****}
41 { This procedure sends a command to an external ARINC 429 channel.
42   Input : ARINC 429 word
43   Output: ARINC 429 word on 429 channel}
44 {*****}
45
46
47 Procedure CloseAr429;
48 {*****}
49 { This procedure makes sure, the computer and peripherals are in their
50   starting state.
51   Input : -
52   Output: -}
53 {*****}
54
55

```

```

56 Implementation
57
58 Uses MIASglob, Miscell, com_4; {comdisc}
59
60 Var
61     mlsport           :      Byte;
62
63
64 Procedure InitAR429;
65
66 Var
67     setupfile      :      Text;
68     title,
69     line           :      String;
70     varname,
71     value          :      String;
72     code           :      Integer;
73
74 Begin
75     MLSPort := 3; { MLSPort is com 3}
76
77     OpenConfigRead( setupfile, MIAScfigname);
78     Repeat { find AR429COMM part of
79             config file}
80         Readln( setupfile, title);
81     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'AR429COMM'));
82
83     If Not Eof( setupfile) { if there is more in file}
84     Then Repeat
85         Readln( setupfile, line); { get a line}
86         Convert( line, varname, value); { extract the variable name
87                                         and value}
88         If ( varname = 'MLSPORT')
89         Then Val( value, mlsport, code);
90         Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
91         { repeat until end of file}
92     CloseConfig( setupfile);
93     { initialise comports for
94       communication with Engine}
95     Setupcomport( mlsport, Ord( B2400), 8, Ord( None), 1);
96     { empty receive and trans-
97       mit buffers}
98     Emptybuffer( mlsport, True);
99     { set interrupt vectors}
100    Installint( mlsport); { save old interrupt}
101 End;
102
103
104 Procedure GetAr429word( Var ar429word: AR429wordtype; Var NoWord: Boolean;
105                        Var a_label: Byte);
106
107 Var
108     x,
109     y      :      Byte;
110     dum    :      Byte;

```

```

111 bitsum,
112 weight      :      Byte;
113
114 Begin
115   If Charsinbuff( mlsport) < 4          { if not enough bytes in the}
116   Then Begin                            { buffer than deliver a clean}
117       For x:= 1 To 32 Do                { AR429 word and exit}
118         Ar429word[ x]:= 0;
119         NoWord:= True;
120         a_label:= 0;
121         Exit;
122     End;
123
124   For x:= 1 To 4 Do
125   Begin
126       dum:= Byte( GetCharbuff( mlsport)); { take a byte from the buffer}
127       For y:= 1 To 8 Do
128         Ar429word[ (x-1) * 8 + (9 - y)]:= (dum Shr ( y -1)) And $01;
129         { take the bits from a byte
130         from the buffer}
131     End;
132
133     bitsum:= 0;                          { check the parity of the}
134     For x:= 1 To 31 Do                   { arinc word. It is odd }
135       bitsum:= bitsum + Ar429word[x];    { parity. Bit 32 is the }
136     bitsum:= bitsum Mod 2;               { paritybit.}
137
138     If bitsum = Ar429word[ 32]
139     Then Begin                            { if bad parity, clear the}
140         For x:= 1 To 32 Do                { AR429 word and exit}
141           Ar429word[ x]:= 0;
142           a_label:=0;
143           NoWord:= True;
144           Exit;
145         End
146     Else Begin
147         NoWord:= False;                   { if the parity was ok,}
148         { the NoWord becomes False}
149         a_label:= 0;                       { determine the label}
150         weight:= 128;                      { for the weights of the bits}
151         For x:= 1 To 8 Do
152         Begin
153             a_label:= a_label + Ar429word[x] * weight;
154             weight:= weight Div 2;
155         End;
156     End;
157 End;
158
159
160 Procedure SendAr429word;
161
162 Begin
163 End;
164
165
166 Procedure CloseAr429;
167
168 Var
169   setupfile      :      Text;
170   Value          :      String;
171
172 Begin
173   Removeint( mlsport);
174
175   OpenConfigWrite( setupfile, MIAScfgname);
176   Writeln( setupfile, 'AR429COMM');
177
178   Str( mlsport, value);
179   Writeln( setupfile, '#9' mlsport = ', value, ';');
180
181   CloseConfig( setupfile);
182 End;
183
184
185 End.
```

```

1 {$A+,B-,D+,E+,F-,G+,I+,L+,N+,O-,R+,S+,V+,X-}
2 {$M 16384,0,655360}
3 Unit ADW;
4
5 (*****) interface (*****
6
7 uses
8   Intrap;
9
10 const
11   BufferLengte = 1024;
12   TO_Tijd     = 50000; {timeout tijd}
13
14 type
15   FifoOBJ = object
16     Kop, Staart      : word;
17     Buffer           : array [0..BufferLengte] of byte;
18     function FIFOfull : boolean;
19     function FIFOempty : boolean;
20     function GetFIFO   : byte;
21     procedure ResetFIFO;
22     procedure PutFIFO (Data : byte);
23   end;
24
25 var
26   FIFO      : FifoOBJ;
27   TimeOut   : boolean;
28
29 procedure InitKaartAdres (Adres : word);
30 procedure ProgTrigFunktie (TrigNum : byte; Funktie : byte);
31 procedure ResetTrigger   (TrigNum : byte);
32 procedure ResetTriggers;
33 procedure IRQ_Aan;
34 procedure IRQ_Uit;
35 procedure ResetIntLatch;
36 procedure Install_ADW_Int;
37 procedure Remove_ADW_Int;
38 procedure KiesIRQ (IRQ : byte);
39
40 function TrigFunktie (TrigNum : byte) : byte;
41
42 (*****) implementation (*****
43
44 uses
45   Crt, DOS;
46
47 const
48   MaxFunkties = 9;
49   IRQ_Bit     = $80; {aan/uit hardware-interrupt, alleen funktie 1}
50   FE_Bit      = $80; {aan/uit funktie, voor funkties 2 t/m 9}
51   KaartAdres : word = $280;
52 (*
53 Adres van een triggerfunctie is gelijk aan het kaartadres plus het
54 trigger-funktienummer min 1.
55 *)

```

```

56 oFunktie_FF = 9; {offset t.o.v. kaartadres}
57 oData_FF    = 10; {offset t.o.v. kaartadres}
58 oReset      = 11; {offset t.o.v. kaartadres}
59 (*
60 Bitmasker voor uitlezen Funktie_FlipFlop.
61 *)
62 mFunktie     = $1F; {functiebits}
63 mFunkPar     = $60; {pariteitbits van funktie}
64 mKlok        = $40; {klok van schuifregister}
65 mPreamble    = $80; {indikatie aanwezigheid gewenste preamble}
66
67 var
68   Trigger : array [1..MaxFunkties] of byte;
69   IRQlijn : byte;
70
71 procedure ResetIntLatch;
72
73 begin
74   Port [KaartAdres+oReset] := 0;
75 end;
76
77 procedure KiesIRQ (IRQ : byte);
78
79 begin
80   if (IRQ >= 3) and (IRQ <= 7)
81   then IRQlijn := IRQ;
82 end;
83
84 (===== INTERRUPT PROCEDURE =====)
85
86 procedure ADW_Handler; interrupt;
87 (*
88 Er is een preamble gedetekteerd. Schakel de IRQ-lijn af om eventuele
89 triggering op data te voorkomen. Lees vervolgens de ontvangen funktiebits
90 uit de Data_FlipFlop en bepaal hieruit hoeveel bits nog zullen volgen.
91 *)
92 const
93   L_BasicDataWord = 32;
94   L_AuxDataWord   = 76;
95
96 var
97   Funktie : word;
98   Data    : word;
99   KlokTeller : word;
100  Kloknivo : word;
101  Bitteller : word;
102  Klaar    : boolean;
103  TO_Teller : longint;
104  MaxBits  : word;
105
106 begin
107   DisableInterrupt;

```

```

111 Funktie := Port [KaartAdres+oFunktie_FF];
112 KlokNivo := (Funktie and mKlok);           {onthoud kloknivo}
113 Data := Port [KaartAdres+oData_FF];
114
115 case (Funktie and mFunktie) of
116   $0A, $1E, $05, $11, $1B, $18 : MaxBits := L_BasicDataWord;
117   { basic data woorden 1 t/m 6 }
118   $07, $15, $0F : MaxBits := L_AuxDataWord;
119   { auxiliary data woorden A t/m C }
120   else MaxBits := 0
121 end; {case}
122
123 Timeout := false;
124 TO_Teller := TO_Tijd;
125
126 if MaxBits > 0
127 then begin
128   with Fifo do
129   begin
130     if not FifoFull
131     then PutFifo (Funktie and mFunktie); {functie identifikatie > fifo}
132
133     if not FifoFull
134     then PutFifo (Data); {volgend byte > fifo}
135
136     repeat {herhalen tot kloknivo laag is}
137       repeat {herhalen tot nivo-verandering van klok}
138         Funktie := Port [KaartAdres+oFunktie_FF];
139       (*
140         if TO_Teller > 0 then Dec (TO_Teller);
141         Timeout := TO_Teller = 0;
142       *)
143     until (KlokNivo <> (Funktie and mKlok)) or Timeout;
144     KlokNivo := (Funktie and mKlok); {verwerk verandering}
145     until (KlokNivo = 0) or Timeout;
146   (*
147   Om de volgende 8 bits in de Data_Flipflop te krijgen moeten 8 klokpulsen
148   geteld worden. Daarna kunnen ze uitgelezen worden en naar de fifo gestuurd
149   worden.
150   *)
151   KlokTeller := 21; {aantal reeds ontvangen klokpulsen}
152   Klaar := false;
153
154   while not (Klaar or Timeout) do
155   begin
156     BitTeller := 1;
157     repeat {herhalen tot volledig byte is ingeschoven}
158       repeat {herhalen tot kloknivo laag is}
159         repeat {herhalen tot nivo-verandering van klok}
160           Funktie := Port [KaartAdres+oFunktie_FF];
161         (*
162         if TO_Teller > 0 then Dec (TO_Teller);
163         Timeout := TO_Teller = 0;
164         *)
165       *)

```

```

166     until (KlokNivo <> (Funktie and mKlok))( or Timeout);
167     KlokNivo := (Funktie and mKlok); {verwerk verandering}
168
169     until (KlokNivo = 0) {or Timeout};
170     Inc (BitTeller); {er is een bit ingeschoven}
171     Inc (KlokTeller);
172     Klaar := KlokTeller = MaxBits; {test op einde bitstroom}
173
174     until (BitTeller mod 8 = 0) or {volledig byte ingeschoven of}
175     Klaar {or Timeout}; {maximum aantal bits ontvangen}
176
177     if not FifoFull {schrijf byte naar fifo}
178     then PutFifo (Port [KaartAdres+oData_FF]);
179   end;
180
181   if not FifoFull then PutFifo ($AA);
182   if not FifoFull then PutFifo ($55);
183 end; {with Fifo do}
184 end
185
186 else begin {onjuiste triggering, wacht tot oorzaak weg is}
187   repeat
188     Funktie := Port [KaartAdres+oFunktie_FF];
189
190     if TO_Teller > 0 then Dec (TO_Teller);
191     Timeout := TO_Teller = 0;
192
193     until (Funktie and mPreamble = 0) or Timeout;
194   end;
195
196   ResetIntLatch; {reset interrupt latch}
197   EnableInterrupt; {interrupts weer toegestaan}
198   ResetInterrupt; {reset hardware interrupt chip}
199 end;
200
201
202 (===== FIFO PERIKELEN =====)
203
204 procedure FifoOBJ.ResetFIFO;
205
206 begin
207   DisableInterrupt;
208   Kop := 0;
209   Staart := 0;
210   EnableInterrupt;
211 end;
212
213
214 function FifoOBJ.FIFOempty : Boolean;
215
216 begin
217   DisableInterrupt;
218   FIFOempty := Kop = Staart;
219   EnableInterrupt;
220 end;

```



```

221
222
223 function FifoOBJ.GetFIFO : byte;
224
225 begin
226   DisableInterrupt;
227   GetFIFO := Buffer [Staat];
228   Inc (Staat);
229   if Staat = BufferLengte then Staat := 0;
230   EnableInterrupt;
231 end;
232
233
234 procedure FifoOBJ.PutFIFO (Data : byte);
235
236 begin
237   Buffer [Kop] := Data;
238   Inc (Kop);
239   if Kop = BufferLengte then Kop := 0;
240 end;
241
242
243 function FifoOBJ.FIFOfull : Boolean;
244
245 begin
246   if Kop = BufferLengte - 1
247   then FIFOfull := 0 = Staat
248   else FIFOfull := (Kop + 1) = Staat;
249 end;
250
251 {===== KAART INSTELLEN =====}
252
253 procedure InitKaartAdres (Adres : word);
254
255 begin
256   KaartAdres := Adres and $03F0; {kaart beslaat 16 adressen}
257 end;
258
259
260 procedure ProgTrigFunktie (TrigNum : byte; Funktie : byte);
261
262 begin
263   if TrigNum <= MaxFunkties
264   then begin
265     Funktie := Funktie and (mFunktie or mFunkPar);
266
267     if TrigNum = 1
268     then Trigger [TrigNum] := Trigger [TrigNum] and not
269                          (mFunktie or mFunkPar) or
270                          Funktie {IRQ-bit ontzien}
271
272     else Trigger [TrigNum] := Funktie or FE_Bit;
273
274     Port [KaartAdres+TrigNum-1] := Trigger [TrigNum];
275   end;
276 end;
277
278
279 function TrigFunktie (TrigNum : byte) : byte;
280
281 begin
282   TrigFunktie := Trigger [TrigNum];
283 end;
284
285
286 procedure ResetTrigger (TrigNum : byte);
287
288 begin
289   if TrigNum <= MaxFunkties
290   then begin
291     if TrigNum = 1
292     then Trigger [TrigNum] := Trigger [TrigNum] or $7F {IRQ-bit ontzien}
293     else Trigger [TrigNum] := $7F;
294     Port [KaartAdres+TrigNum-1] := Trigger [TrigNum];
295   end;
296 end;
297
298
299 procedure ResetTriggers;
300
301 var
302   n : byte;
303
304 begin
305   for n := 1 to MaxFunkties do ResetTrigger (n);
306 end;
307
308
309 procedure IRQ_Aan;
310
311 begin
312   Trigger [1] := Trigger [1] or IRQ_bit;
313   Port [KaartAdres] := Trigger [1];
314 end;
315
316
317 procedure IRQ_Uit;
318
319 begin
320   Trigger [1] := Trigger [1] and not IRQ_bit;
321   Port [KaartAdres] := Trigger [1];
322 end;
323
324 {=====}
325
326 procedure Install_ADW_Int;
327
328 begin
329   if IRQlijn <> 0
330   then begin

```

Page 4, listing of ADW.PAS, date is 18-02-93, file date is 18-02-93, size is 9014 bytes.

```
331 SetInterrupt (IRQlijn, ON, Addr (ADW_Handler));
332 ResetTriggers;
333 IRQ_Aan;
334 ResetIntLatch;           {reset interrupt latch}
335 end;
336 end;
337
338
339 procedure Remove_ADW_Int;
340
341 begin
342 SetInterrupt (IRQlijn, OFF, Addr (ADW_Handler));
343 IRQ_Uit;
344 ResetIntLatch;           {reset interrupt latch}
345 end;
346
347
348 (===== INITIALISATIE =====)
349
350 begin
351 IRQlijn := 0;
352 Fifo.ResetFifo;
353 IRQ_Uit;
354 end.
```

```

1 {=====}
2 {
3 {          unit:  INTERRUPT HANDLING          }
4 {          =====}
5 {
6 { File: INTRUPT.PAS
7 {
8 { This unit contains functions to handle interrupts
9 {
10 {
11 {          feb 88
12 {=====}
13
14
15 { $D-,R-,S-,V-}
16 UNIT Inrupt;
17
18
19 INTERFACE
20 {*****}
21
22 USES Dos;
23
24 CONST
25   On = true;
26   Off = false;
27
28
29 PROCEDURE EnableInterrupt; Inline ($FB);
30 PROCEDURE DisableInterrupt; Inline ($FA);
31 PROCEDURE ResetInterrupt; Inline ($B0/$20/$E6/$A0/$E6/$20);
32 PROCEDURE SetInterrupt (IRQ: byte; ON: boolean; VEKTOR: pointer);
33 PROCEDURE ExeOldInterrupt (IRQ: byte);
34
35
36 IMPLEMENTATION
37 {*****}
38
39 const
40   IntNrs = 15;
41   Int : array [0..IntNrs] of byte = ($08,$09,$0A,$0B,$0C,$0D,$0E,$0F,
42                                     $70,$71,$72,$73,$74,$75,$76,$78);
43 var
44   IRQPTR : array [0..IntNrs] of pointer;
45   n      : byte;
46
47
48 {-----}
49 { PROCEDURE EnableInterrupt; Inline ($FB);
50 {-----}
51 { in: -
52 { out: -
53 { rem: This is the machine instruction STI Set Interrupt Flag
54 {-----}
55
56
57
58 {-----}
59 { PROCEDURE DisableInterrupt; Inline ($FA);
60 {-----}
61 { in: -
62 { out: -
63 { rem: This is the machine instruction CLI Clear Interrupt Flag
64 {-----}
65
66
67
68 {-----}
69 { PROCEDURE ResetInterrupt; Inline ($B0/$20/$E6/$A0/$E6/$20);
70 {-----}
71 { in: -
72 { out: -
73 { rem: This is a macro for the following machine instructions:
74 {      MOV AL,20h
75 {      OUT 0A0h,AL clear slave interrupt unit 8259
76 {      OUT 020h,AL clear master interrupt unit 8259
77 {      The master interrupt handles hardware interrupts IRQ0..IRQ7
78 {      The slave interrupt handles hardware interrupts IRQ8..IRQ15
79 {      The slave chip is NOT present into IBM-XT
80 {      ResetInterrupt must be the last instruction in a hardware
81 {      interrupt serving procedure.
82 {-----}
83
84
85
86
87 PROCEDURE SetInterrupt (IRQ: byte; ON: boolean; VEKTOR: pointer);
88
89 {-----}
90 { in:  IRQ - the number 0..15 of the interrupt request line
91 {      ON - TRUE: enable hardware interrupt IRQ
92 {      FALSE: disable hardware interrupt IRQ
93 { out: -
94 { rem: This call must be used to enable user interrupts at initialzing
95 {      time, and must be used as well to disable user interrupts
96 {      before ending the program. The following hardware is present:
97 {
98 {   Line   Interrupt   PC - XT           AT - PS/2(50,60)
99 {   -----
100 {   IRQ0    Int $08      Timer              Timer
101 {   IRQ1    Int $09      Keyboard             Keyboard
102 {   IRQ2    Int $0A      Vert. retrace EGA     cascade interrupts 8..15
103 {   IRQ3    Int $0B      COM2                  COM2
104 {   IRQ4    Int $0C      COM1                  COM1
105 {   IRQ5    Int $0D      Fixed Disk adapter    Printer 2
106 {   IRQ6    Int $0E      Diskette adapter      Diskette adapter
107 {   IRQ7    Int $0F      Printer adapter       Printer 1
108 {
109 {   IRQ8    Int $70      --                    Real time clock chip
110 {   IRQ9    Int $71      --                    Reserved, NetWork

```

```

111 { IRQ10      $72      --          COM3          }          166      inline ($9C);          { PushF }
112 { IRQ11      $73      --          COM4          }          167      inline ($FF/$1E/Ptr); { Call Far [oude interrupt] }
113 { IRQ12      $74      --          Reserved (PS/2: Mouse) }          168      end;
114 { IRQ13      $75      --          80x87 co processor }          169      end;
115 { IRQ14      $76      --          Fixed Disk adapter }          170 { $F- }
116 { IRQ15      $78      --          Reserved          }          171
117 {-----}          172
118          173 BEGIN
119 VAR          174      for n := 0 to IntNrs do IRQPTR [n] := nil; { maak alle pointers nil }
120 NewInt : word;          175 END.
121 Mask : record          176
122     case boolean of          177
123     TRUE : (All          : word);          178
124     FALSE: (LowByte,HighByte: byte);
125     end;
126
127 BEGIN
128 DisableInterrupt;          { no interrupts while changing }
129 NewInt := 1 SHL IRQ;          { set bit according to value }
130 Mask.HighByte := Port[$A1];          { all enabled slave interrupts }
131 Mask.LowByte := Port[$21];          { all enabled master interrupts }
132
133 if ON and          { IF NEW INTERRUPT ENABLED and }
134 (IRQPTR [IRQ] = nil)          { not enabled before }
135 then begin
136     NewInt := NewInt XOR $FF;          { swap all bits }
137     Mask.All := Mask.All AND NewInt;          { clear the new bit in Mask }
138     GetIntVec (Int [IRQ], IRQPTR [IRQ]);          { onthoud huidige pointer }
139     SetIntVec (Int [IRQ], Vektor);          { pointer naar programma }
140 end
141 else begin          { IF INTERRUPT DISABLED }
142     if IRQPTR [IRQ] <> nil          { and interrupt was saved }
143     then begin
144         SetIntVec (Int [IRQ], IRQPTR [IRQ]);          { herstel originele pointer }
145         IRQPTR [IRQ] := nil;          { geen interrupt meer bewaard }
146         Mask.All := Mask.All OR NewInt;          { set the new bit in Mask }
147     end;
148 end;
149
150 Port[$A1] := Mask.HighByte;          { set new slave mask }
151 Port[$21] := Mask.LowByte;          { set new master mask }
152 EnableInterrupt;          { interrupts back on }
153 END; { SetInterrupt }
154
155
156 { $F+ }
157 PROCEDURE ExeOldInterrupt (IRQ: byte);
158
159 var
160     Ptr : pointer;
161
162 begin
163     if IRQPTR [IRQ] <> nil
164     then begin
165         Ptr := IRQPTR [IRQ];

```

```

1 Unit miscell;
2
3 Interface
4
5 Const
6     zeros      =      '00000000000000000000000000000000';
7
8 Type
9     timetype    =      Record
10         year,
11         month,
12         day,
13         hour,
14         minute,
15         sec,
16         sec100   :      Word;
17
18     End;
19
20 Procedure Convert( line: String; Var varname, value: String);
21 {*****}
22 { This procedure converts a line from the MIAS.CFG file. It outputs
23   the variable name and the value in upcase. A line should begin with one
24   or more spaces, then the variable name, followed by '=', terminated by
25   the value in ascii and a semicolon.
26   Input : line from CFG file
27   Output: variablename
28         value}
29 {*****}
30
31 Procedure OpenConfigRead( Var setupfile: Text; filename: string);
32 {*****}
33 { This procedure opens the config-file for reading.
34   Input : -
35   Output: file of Text}
36 {*****}
37
38
39 Procedure CloseConfig( Var setupfile: Text);
40 {*****}
41 { This procedure closes the config-file.
42   Input : file of Text
43   Output: -}
44 {*****}
45
46
47 Procedure OpenConfigWrite( Var setupfile: Text; filename: String);
48 {*****}
49 { This procedure appends the config-file for writing.
50   Input : filename
51   Output: file of Text}
52 {*****}
53
54
55 Procedure OpenConfigWriteFirst( Var setupfile: Text; filename: String);
56 {*****}
57 { This procedure opens the config-file for writing.
58   Input : filename
59   Output: file of Text}
60 {*****}
61
62
63 Procedure Date_and_Time( Var time: timetype);
64 {*****}
65 { Fills the input variable with the current system date and time.
66   Input : -
67   Output: system date and time}
68 {*****}
69
70
71 Procedure ErrorTime( Var time: timetype);
72 {*****}
73 { This procedure sets the time variable to 0,0,0,0 to indicate an error
74   in the data it represents.
75   Input : -
76   Output: 0,0,0,0 in time}
77 {*****}
78
79
80 Procedure Addtime( t1, t2: timetype; Var sum: timetype);
81 {*****}
82 { This procedure adds two times. The result is corrected for leap-years.
83   Input : t1, t2; times to be added
84   Output: sum; the addition result}
85 {*****}
86
87
88 Function Later( t1, t2: timetype): Boolean;
89 {*****}
90 { This function becomes True if t1 is later than or equal to t2. Leap years
91   are accounted for.
92   Input : t1, t2, times to be compared
93   Output: Boolean.}
94 {*****}
95
96
97 Function FileExist( filename: String): Boolean;
98 {*****}
99 { This function checks the current directory for a file specified by
100   filename. If the file is present, true is output.
101   Input : filename
102   Output: true/false}
103 {*****}
104
105
106 Implementation
107
108 Uses Dos;
109
110

```

```

111 Procedure Convert( line: String; Var varname, value: String);
112
113 Begin
114     varname:= '';
115     value:= '';
116
117     If (line[1] <> ' ') And ( line[1] <> #9){ exit when no leading space}
118     Then Begin
119         Exit;
120     End;
121
122     While (( line[1] = ' ') Or ( line[1] = #9)) And
123         ( Length( line) > 1) Do { skip leading spaces}
124         line:= Copy( line, 2, Length( line) -1);
125     If ( Length( line) = 0) { line too short}
126     Then Exit;
127
128     While ( line[ 1] <> ' ') And ( line[ 1] <> '=') And
129         ( Length( line) > 1) Do { get varname}
130     Begin
131         varname:= varname + Ucase( line[1]);
132         line:= Copy( line, 2, Length( line) -1);
133     End;
134     If ( Length( line) = 0) { line too short}
135     Then Exit;
136
137     While ( line[ 1] <> '=') And ( Length( line) > 1) Do
138         { search for '='}
139         line:= Copy( line, 2, Length( line) -1);
140     If ( Length( line) < 2)
141     Then Exit;
142
143     line:= Copy( line, 2, Length( line) -1);{ skip '='}
144     While (( line[1] = ' ') Or ( line[1] = #9)) And
145         ( Length( line) <> 0) Do { skip leading spaces}
146         line:= Copy( line, 2, Length( line) -1);
147     If ( Length( line) = 0)
148     Then Exit;
149
150     While ( line[ 1] <> ';') And ( Length( line) > 1) Do
151         { get value}
152     Begin
153         value:= value + line[1];
154         line:= Copy( line, 2, Length( line) -1);
155     End;
156 End;
157
158
159 Procedure OpenConfigRead( Var setupfile: Text; filename: string);
160
161 Begin
162     If FileExist( filename)
163     Then Begin
164         Assign( setupfile, filename);
165         Reset( setupfile);
166
167         Else Begin
168             Assign( setupfile, filename);
169             Rewrite( setupfile);
170             Writeln( setupfile); { put some dummy load in file}
171             Close( setupfile);
172             Reset( setupfile);
173         End;
174
175 End;
176
177
178 Procedure CloseConfig( Var setupfile: Text);
179
180 Begin
181     If (TextRec( setupfile). mode = fmInput) Or
182         (TextRec( setupfile). mode = fmOutput) Or
183         (TextRec( setupfile). mode = fmInOut)
184     Then Close( setupfile);
185 End;
186
187
188 Procedure OpenConfigWrite( Var setupfile: Text; filename: String);
189
190 Begin
191     If FileExist( filename)
192     Then Begin
193         Assign( setupfile, filename);
194         Append( setupfile);
195     End
196     Else Begin
197         Assign( setupfile, filename);
198         Rewrite( setupfile);
199     End;
200 End;
201
202
203 Procedure OpenConfigWriteFirst( Var setupfile: Text; filename: String);
204
205 Begin
206     Assign( setupfile, filename);
207     Rewrite( setupfile);
208 End;
209
210
211 Procedure Date_and_Time( Var time: timetype);
212
213 Var
214     dum : Word;
215 Begin
216     With time Do
217     Begin
218         Getdate( year, month, day, dum);
219         Gettime( hour, minute, sec, sec100);
220     End;

```

```

221 End;
222
223
224 Procedure ErrorTime( Var time: timetype);
225
226 Begin
227     With time Do
228     Begin
229         year:= 0;
230         month:= 0;
231         day:= 0;
232         hour:= 0;
233         minute:= 0;
234         sec:= 0;
235         sec100:= 0;
236     End;
237 End;
238
239
240 Procedure Addtime( t1, t2: timetype; Var sum: timetype);
241
242 Begin
243     sum.sec100:= t1.sec100 + t2.sec100;
244     If sum.sec100 > 99 Then Begin
245         sum.sec100:= 99;
246         sum.sec:= 1;
247     End
248     Else sum.sec:= 0;
249
250     sum.sec:= sum.sec + t1.sec + t2.sec;
251     If sum.sec > 59 Then Begin
252         sum.sec:= 59;
253         sum.minute:= 1;
254     End
255     Else sum.minute:= 0;
256
257     sum.minute:= sum.minute + t1.minute + t2.minute;
258     If sum.minute > 59 Then Begin
259         sum.minute:= 59;
260         sum.hour:=1;
261     End
262     Else sum.hour:= 0;
263
264     sum.hour:= sum.hour + t1.hour + t2.hour;
265     If sum.hour > 23 Then Begin
266         sum.hour:= 23;
267         sum.day:=1;
268     End
269     Else sum.day:= 0;
270
271     sum.day:= sum.day + t1.day + t2.day;
272     sum.month:= 0;
273     Case t1.month Of
274     1,3,5,7,8,10,12: If sum.day > 31
275     Then Begin

```

```

276 sum.day:= 31;
277 sum.month:= 1;
278 End;
279 4,6,9,11: If sum.day > 30
280 Then Begin
281     sum.day:= 30;
282     sum.month:=1;
283 End;
284 2: If ( t1.year Mod 4 = 0) And ( sum.day > 29)
285 Then Begin
286     sum.day:= 29;
287     sum.month:=1;
288 End
289 Else If sum.day > 28
290 Then Begin
291     sum.day:= 28;
292     sum.month:= 1;
293 End;
294 End;
295
296 sum.month:= sum.month + t1.month + t2.month;
297 If sum.month > 12
298 Then Begin
299     sum.month:= 12;
300     sum.year:=1;
301 End
302 Else sum.year:= 0;
303
304 sum.year:= sum.year + t1.year + t2.year;
305 End;
306
307
308 Function Later( t1, t2: timetype): Boolean;
309
310 Begin
311 If t1.year > t2.year
312 Then later:= True
313 Else If t1.year = t2.year
314
315 Then If t1.month > t2.month
316 Then later:= True
317 Else If t1.month = t2.month
318
319 Then If t1.day > t2.day
320 Then later:= True
321 Else If t1.day = t2.day
322
323 Then If t1.hour > t2.hour
324 Then later:= True
325 Else If t1.hour = t2.hour
326
327 Then If t1.minute > t2.minute
328 Then later:= True
329 Else If t1.minute = t2.minute
330

```

```
331             Then If t1.sec > t2.sec
332                 Then later:= True
333                 Else If t1.sec = t2.sec
334
335                     Then If t1.sec100 >= t2.sec100
336                         Then later:= True
337                         Else later:= False
338
339                     Else later:= False
340                 Else later:= False
341             Else later:= False
342         Else later:= False
343     Else later:= False
344 Else later:= False;
345 End;
346
347
348 Function FileExist( filename: String): Boolean;
349
350 Var
351     infile      :      File Of Byte;
352
353 Begin
354     {$I-}
355     Assign( infile, filename);
356     Reset( infile);
357
358     If IOResult <> 0
359     Then FileExist:= False
360     Else Begin
361         FileExist:= True;
362         Close( infile);
363     End;
364     {$I+}
365 End;
366
367 End.
```



```

1 UNIT Com_4;
2
3 { The original program is called:}
4 { Comm_TP4.PAS Ver. 1.50 - RS-232 Support for IBM Compatibles }
5 { (c) Copyright, 1989 }
6 { Kevin R. Bulgrien }
7 { October, 1989 }
8 {
9 { See the accompanying file COMM_TP4.DOC for specific information regarding
10 { the distribution policies and usage information for this source code file. }
11 {
12 { Written by: Kevin R. Bulgrien Version 1.50 completed 11/13/89 }
13 {
14 { Contact at: LeTourneau University LeTourneau University BBS }
15 { Microcomputer Services 2400/1200/300 Baud }
16 { P.O. Box 7001 (214) 237-2742 }
17 { Longview, TX 75607 }
18 {
19 { This program works with Turbo Pascal 4.0 and 5.x. See Comm_TP4.DOC for the }
20 { instructions. Comm_TP3, by the same author, works under Turbo Pascal 3.0, }
21 { and Comm_TC2 works with Turbo C. Upcoming is a Turbo Assembler Comm_TA1. }
22 {
23 { This software directly accesses the 8250 UART as well as the 8259 interrupt }
24 { controller hardware. Though they are IBM's standard, it is possible that }
25 { some manufacturers could use different hardware to perform these functions. }
26 {
27 { This unit was adapted and expanded by R.C. Meijer}
28 {
29 { Functions Getcharbuff, lookbuff, restore_buff and charsinbuff added by }
30 { Comments added in Interface section }
31 {
32 { NOTE: THE INTERRUPT NUMBER FOR COM3 IS CHANGED TO IRQ5 AND}
33 { THE INTERRUPT NUMBER FOR COM4 IS CHANGED TO IRQ7 !!!!!}
34 { IT IS NOW POSSIBLE TO RUN COM1, COM2, COM3 AND COM4}
35 { AT THE SAME TIME}
36 { ON THE I/O CARD, THE INTERRUPT FOR LPT1 AND LPT2 SHOULD BE DISABLED}
37 { A SPECIAL COMPORT CARD SHOULD BE USED PROVIDING THE APPROPRIATE{
38 { INTERRUPTS. }
39 {
40 {$S-} { Interrupt handlers should not be compiled with stack checking enabled }
41 { -Else the system may crash! }
42 {$DEFINE ErrorChecking} {-Enable/Disable Error Check }
43 {$DEFINE NoMessageCode} {-Enable/Disable Error Msgs }
44 {$DEFINE FWriteCOM} {-WriteCOM is a Func or Proc }
45 {
46 INTERFACE
47 {
48 USES DOS, CRT;
49 {
50 CONST
51 MaxPorts = 4; {-Max # of COM ports to use }
52 MaxInSize = 1000; {-Maximum input buffer size }
53 MaxOutSize = 255; {-Maximum output buffer size }
54 {-These constants are used }
55 {-to make the code readable. }
56
57
58
59
60
61
62
63 THR = 0;
64 RHR = 0;
65 DLL = 0;
66 IER = 1;
67 DLM = 1;
68 IIR = 2;
69 LCR = 3;
70 MCR = 4;
71 LSR = 5;
72 MSR = 6;
73
74 { The following declarations are crucial to the operation of this program. }
75 { I would advise you not to change the information unless you are sure you }
76 { know what you are doing. See the .DOC file for further information. For }
77 { standard MaxPorts settings of 1 - 4, move the comment bracket as needed. }
78 { Improper setting of the IRQNmb array may cause a system crash! }
79
80 COMNmb : ARRAY [1..MaxPorts] OF BYTE = ( 1, 2, 3, 4 );
81 {-Define the COM port number }
82 COMPort : ARRAY [1..MaxPorts] OF WORD = ( $03F8, $02F8, $03E8, $02E8 );
83 {-Base addresses 8250 Regs }
84 IRQNmb : ARRAY [1..MaxPorts] OF BYTE = ( 4, 3, 5, 7 );
85 {-IRQ numbers of the ports }
86 ChainInt : ARRAY [1..MaxPorts] OF BOOLEAN = ( FALSE, FALSE, FALSE, FALSE );
87 {-When port interrupt done, }
88 {-jump to OldIntVector [x]? }
89
90
91
92 Type
93 BaudType = (B110,B150,B300,B600,B1200,B2400,B4800,B9600,B19200,B38400);
94 {-Baud rates supported }
95 ParityType = (None, Odd, Null, Even, MarkOff, Mark, SpaceOff, Space);
96 {-Parity types supported }
97 ProcNameType = STRING [20]; {-Used by error checking code}
98
99 VAR
100 Framing, Overrun, Parity, Break : ARRAY [1..MaxPorts] OF WORD;
101 {-Port error counters }
102 OutBuffer : ARRAY [1..MaxPorts, 0..MaxOutSize] OF BYTE;
103 {-Port output buffers }
104 InBuffer : ARRAY [1..MaxPorts, 0..MaxInSize] OF BYTE;
105 {-Port input buffers }
106 CTS, DSR, RI, CD : ARRAY [1..MaxPorts] OF BOOLEAN;
107 {-Port input line status }
108 OutHead, OutTail : ARRAY [1..MaxPorts] OF WORD;{-Output buffer pointers }
109 InHead, InTail : ARRAY [1..MaxPorts] OF WORD; {-Input buffer pointers }
110 IntInstalled : ARRAY [1..MaxPorts] OF BOOLEAN; {-TRUE if interrupt in place }

```

```

111 DTR_RTS : ARRAY [1..MaxPorts] OF BOOLEAN;      {-Allowed to alter DTR/RTS? }
112 ErrorCode, ErrorPort : BYTE;                  {-Error type code & the port }
113 {$IFDEF NoMessageCode}                        { which had the error.      }
114   ErrMsgX, ErrMsgY : BYTE;                      {-Error message coordinates }
115   ShowMessages : BOOLEAN;                       {-FALSE disables the error  }
116 {$ENDIF}                                       { messages/TRUE enables them.}
117 cnter:Byte;
118
119 PROCEDURE DisableInts; INLINE ($FA);             {-Disable hardware interrupts}
120 {*****}
121 { This procedure disables all interrupts. It is used, so that some
122   operations, which would give wrong result when interrupted, can work
123   properly.
124                               Input : -
125                               Output: -}
126 {*****}
127
128
129 PROCEDURE EnableInts; INLINE ($FB);              {-Enable hardware interrupts }
130 {*****}
131 { This procedure enables all interrupts. It is used to undo disableints.
132                               Input : -
133                               Output: -}
134 {*****}
135
136
137 PROCEDURE Set_DTR_RTS (Com : BYTE; Status : BOOLEAN);
138 {*****}
139 { This procedure changes the setting of DTR & RTS if the global variable
140   DTR_RTS is set to TRUE. The port and the desired state are passed as
141   parameters. TRUE for on, FALSE for off. Since hardware handshaking
142   requirements vary according to the hardware being used, you may have
143   to rewrite Set_DTR_RTS to accommodate the hardware. Bit 0 controls
144   DTR and bit 1 controls RTS. Writing a 0 to the bit will turn the line
145   on.
146                               Input : comportsnumber
147                               status flag
148                               Output: -}
149 {*****}
150
151
152 PROCEDURE SetupCOMPort (Com, Baud, DataBits, Parity, StopBits : BYTE);
153 {*****}
154 { This procedure is used to program the 8250 UART chip. You can use the
155   UART for serial communications and as a mouse interface.
156                               Input : comportsnumber
157                               baudrate
158                               number of databits
159                               parity
160                               number of stopbits
161                               Output: -}
162 {*****}
163
164
165 PROCEDURE InstallInt (Com : BYTE);

```

```

166 {*****}
167 { This procedure is used to install the interrupts used for the serial
168   communications. For comports 1, the IRQ nr 4 is used and interruptvector
169   $0C. For comports 2, the IRQ nr 3 is used and interruptvector $0B.
170                               Input : comportsnumber
171                               Output: -}
172 {*****}
173
174
175 PROCEDURE RemoveInt (Com : BYTE);
176 {*****}
177 { This procedure is used to uninstall the interrupts. The original inter-
178   rupts are installed again, to insure further operation.
179                               Input : comportsnumber
180                               Output: -}
181 {*****}
182
183
184 PROCEDURE EmptyBuffer (Buffer : BYTE; TrueInFalseOut : BOOLEAN);
185 {*****}
186 { This procedure is used to empty a buffer used for communications port.
187                               Input : buffer
188                               flag to indicate input/output
189                               buffer
190                               Output: -}
191 {*****}
192
193
194 {$IFDEF FWriteCOM}
195 FUNCTION WriteCOM (Com : BYTE; Data : STRING) : BOOLEAN;
196 {$ELSE}
197 PROCEDURE WriteCOM (Com : BYTE; Data : STRING);
198 {$ENDIF}
199 {*****}
200 { Depending on the definition of the environment variable FWRITECOM, the
201   procedure or function is used to transmit a string of data on a certain
202   comports.
203                               Input : comportsnumber
204                               datastring
205                               Output: (-/ flag indicating success)}
206 {*****}
207
208
209 PROCEDURE IWriteCOM (Com : BYTE; Data : STRING);
210 {*****}
211 { This procedure writes a string to the comports specified. It uses inter-
212   rupts to do its job.
213                               Input : comportsnumber
214                               datastring
215                               Output: -}
216 {*****}
217
218
219 FUNCTION ReadCOM (Com : BYTE) : CHAR;
220 {*****}

```

```

221 { This function reads a character from a comport. It is used for polling.
222       Input : comportnumber
223       Output: character}
224 {*****}
225
226
227 FUNCTION TimedReadCOM (Com : BYTE; VAR Data : CHAR) : BOOLEAN;
228 {*****}
229 { This function reads a character from a comport. It wait some time to
230   catch a character. If the character was not caught, the it return a false.
231       Input : comportnumber
232       Output: character from comport
233             flag indicating timed out}
234 {*****}
235
236
237 Function Getcharbuff( comport: Byte): Char;
238 {*****}
239 { This function fetches one character from the buffer that belongs to
240   a comport. It return #00 if there was no character.
241       Input : comportnumber
242       Output: character}
243 {*****}
244
245
246 Function Lookbuff( comport: Byte): Char;
247 {*****}
248 { This function copies one character from the buffer that belongs to a com-
249   port. It is not erased in the buffer. It returns #00 if there was no
250   character.
251       Input : comportnumber
252       Output: character}
253 {*****}
254
255
256 Function Charsinbuff( comport: Byte): Integer;
257 {*****}
258 { This function returns the number of characters in the buffer for one
259   comport.
260       Input : comportnumber
261       Output: number of characters in buffer}
262 {*****}
263
264
265 Procedure Restore_buffer( comport: Byte; line: String);
266 {*****}
267 { This procedure restores a line into the read buffer.
268       Input : comportnumber
269             line to restore
270       Output: restored buffer}
271 {*****}
272
273 IMPLEMENTATION
274
275

```

```

276 VAR
277   OldIntVector : ARRAY [1..MaxPorts] OF POINTER; {-Original COMx int. vectors }
278   IRQMask, Loop : BYTE;
279   ExitSave : POINTER;           {-Saves original ExitProc  }
280   restorebuff  :   Array[1..Maxports] Of String;
281   restoreflag  :   Array[1..Maxports] Of Boolean;
282
283 { This procedure sets the global error identification variables ErrorCode and
284   ErrorPort. It also prints an appropriate error message if the $DEFINE
285   NoMessageCode directive was not present at compile time and if ShowMessages
286   is TRUE at run-time. The error messages are displayed at the current
287   cursor position when ErrMsgX and/or ErrMsgY = 0 (Default). If the calling
288   program needs the messages at a specific location, it must set ErrMsgX and
289   ErrMsgY to the desired screen coordinates. Error messages consist of a
290   beep, the procedure or function in which the error took place, the port
291   number, and a description of the problem encountered. At all times,
292   ErrorCode and ErrorPort can also be used to find error conditions.
293   ErrorCode is 0 for no errors, or a value from 1-5 describing the error.
294   ErrorPort is the handler number which had the problem. These values are
295   valid for the last serial operation requested. }
296
297 PROCEDURE MakeError (Code, Port : BYTE; ProcName : ProcNameType);
298 BEGIN
299   ErrorCode := Code;           {-Set the global error type  }
300   ErrorPort := Port;          { and port variables.      }
301   {$IFDEF NoMessageCode}
302   IF ShowMessages
303   THEN BEGIN
304     IF (ErrMsgX > 0) THEN GOTOXY (ErrMsgX, WHEREY);
305                                     {-Print error messages. A 0  }
306     IF (ErrMsgY > 0) THEN GOTOXY (WHEREX, ErrMsgY);
307                                     { coordinate uses current  }
308     WRITE (ProcName, ' ERROR: ', #7); { cursor position.      }
309     CASE Code OF
310       1 : WRITELN ('Invalid port # ', Port);
311                                     { 1 <= Good Port <= MaxPorts }
312       2 : WRITELN ('Port # ', Port, 'already installed');
313                                     {-Use InstallInt once/port  }
314       3 : WRITELN ('Port # ', Port, 'not installed yet');
315                                     {-RemoveInt w/o InstallInt  }
316       4 : WRITELN ('Timeout writing port # ', Port);
317                                     {-WriteCOM error              }
318       5 : WRITELN ('Timeout reading port # ', Port);
319                                     {-TimedReadCOM error          }
320     END;
321   END;
322   {$ENDIF}
323 END;
324
325 { This function is used to make sure that the requested ports are valid and
326   if the interrupt handlers are properly installed or uninstalled. It calls
327   MakeError to set the global error variables ErrorCode and ErrorPort, and to
328   print error messages. Status should be set to 0 if the port handler should
329   not be installed yet, and 1 if it is supposed to be installed already. Use
330   a status of -1 if the installation status is not critical. ProcName is

```

```

331 used to pass the procedure or function name to MakeError so it can be used
332 in an error message. A TRUE is returned if everything checks out okay,
333 otherwise a FALSE is returned.)
334
335 FUNCTION ValidPort (Port:BYTE; Status:INTEGER; ProcName:ProcNameType):BOOLEAN;
336 BEGIN
337     ErrorCode := 0;                                {-Default of no errors found }
338     IF (Port < 1) OR (Port > MaxPorts)                {-Check requested port # for }
339     THEN MakeError (1, Port, ProcName)               { validity. }
340     ELSE IF (Status >= 0) AND (ORD (IntInstalled [Port]) <> Status)
341     THEN MakeError (2+Status, Port, ProcName);       {-Check port installation }
342     THEN MakeError (2+Status, Port, ProcName);
343     ValidPort := (ErrorCode = 0);                    { state with needed status. }
344     {-Returns TRUE if no errors }
345 END;
346
347 { This procedure changes the setting of DTR & RTS if the global variable
348 DTR_RTS is set to TRUE. The port and the desired state are passed as
349 parameters. TRUE for on, FALSE for off. Since hardware handshaking
350 requirements vary according to the hardware being used, you may have to
351 rewrite Set_DTR_RTS to accommodate the hardware. Bit 0 controls DTR and
352 bit 1 controls RTS. Writing a 0 to the bit will turn the line on.)
353
354 PROCEDURE Set_DTR_RTS (Com : BYTE; Status : BOOLEAN);
355 BEGIN
356     {$IFDEF ErrorChecking}
357     IF NOT ValidPort (Com, -1, 'Set_DTR_RTS') THEN EXIT;
358     {-Optional error trapping }
359     {$ENDIF}
360     IF DTR_RTS [Com]
361     THEN IF Status
362     THEN PORT [COMPort [Com]+MCR] := PORT [COMPort [Com]+MCR] AND $FC
363     ELSE PORT [COMPort [Com]+MCR] := PORT [COMPort [Com]+MCR] OR $03;
364     ELSE PORT [COMPort [Com]+MCR] := PORT [COMPort [Com]+MCR] OR $03;
365     {- dropping DTR and/or RTS. }
366 END;
367
368 { This procedure sets up a selected serial port to use specified parameters.
369 Com specifies the port to set up. The Baud parameter must be in the range
370 0 to 9, and is not range checked, but its maximum valid value is determined
371 by the number of entries in BaudTable. BaudType documents the baud rates
372 supported by BaudTable. ParityType is provided to document the parity
373 settings allowed. Use ORD() to get the correct value to pass: ORD(B110)
374 returns the BYTE that selects 110 baud and ORD(None) gives the value that
375 selects no parity. 1.5 stop bits are used when StopBits = 2 AND DataBits
376 = 5, otherwise StopBits will set the indicated number of stop bits in the
377 range 1 to 2. DataBits may be set with 5 to 8 for the number of data bits
378 to use. Mark parity means that the parity bit is always set to 0. Space
379 parity means that the parity bit is always set to 1. MarkOff, SpaceOff,
380 NONE, & NULL all disable parity but are here for completeness. }
381
382 PROCEDURE SetupCOMPort (Com, Baud, DataBits, Parity, StopBits : BYTE);
383 CONST BaudTable : ARRAY [0..9] OF WORD = ($0417, $0300, $0180, $00C0, $0060,
384 $0030, $0018, $000C, $0006, $0003);
385 { Set baud rate of 8250 when }
386
387 VAR
388     Temporary : BYTE;
389 BEGIN
390     {$IFDEF ErrorChecking}
391     IF NOT ValidPort (Com, -1, 'SetupCOMPort') THEN EXIT;
392     {-Optional error trapping }
393     {$ENDIF}
394     Set_DTR_RTS (Com, FALSE);
395     PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] OR $80;
396     {-Set DLL & DLM active }
397     PORT [COMPort [Com] + DLL] := LO (BaudTable [Baud]);
398     {-Set the baud rate with the }
399     PORT [COMPort [Com] + DLM] := HI (BaudTable [Baud]);
400     {- predefined divisor values. }
401     Temporary := (DataBits - 5) AND $03 OR (((StopBits - 1) SHL 2) AND $04);
402     {-Set data bits, stop bits, }
403     PORT [COMPort [Com] + LCR] := Temporary OR ((Parity SHL 3) AND $38);
404     {- and parity protocol. }
405     Set_DTR_RTS (Com, TRUE);
406 END;
407
408 { These procedures handle all interrupts from the 8250 communications chip.
409 All interrupt types are at least minimally supported. Enough code is
410 present to help you know how to modify the code for your specific
411 requirements. Incoming data is ignored if the buffer is full, otherwise
412 it is placed into the InBuffer circular queue. Data to be transmitted by
413 interrupt is taken from the OutBuffer queue. The transmitter is the only
414 interrupt which has to be manually invoked. Do so by placing the first
415 character of each trans- mission into the THR. It automatically shuts off
416 when all the data in the buffer has been sent. The other interrupt types
417 will take care of themselves once enabled. Modem/Port input lines may be
418 monitored by this handler. BOOLEAN arrays CTS, DSR, RI, and CD always
419 show current status of these lines IF the interrupt handler is active and
420 the Modem Status Change interrupt has been enabled. A TRUE indicates the
421 signal is active. CD and RI are very helpful for modem related programs.
422 Line Status errors are counted. It is up to you add any corrective action.
423 All ports with interrupts pending on the IRQ level which invoked this
424 handler are processed regardless of which port generated the actual
425 interrupt. This is simple to implement, yet it preserves the interrupt
426 priority handling between ports. }
427
428 {$F+}
429 PROCEDURE IntHandler1; INTERRUPT;
430 {-Interrupt handlers MUST be }
431 {- FAR calls. }
432
433 CONST
434     com = 1;
435
436 VAR
437     imr,
438     temp : Byte;
439
440 BEGIN
441     IMR := PORT [$21];
442     {-Backup IMR for later use }
443     PORT [$21] := IMR OR IRQMask;
444     {-Disable Comm_TP4 interrupts}

```

```

441 EnableInts;                                { -Allow other interrupts }
442
443 PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;
444
445                                { -Set THR, RHR & IER active }
446 CASE PORT [COMPort [Com] + IIR] AND $06 OF { -Identify interrupt type }
447 0 : BEGIN
448     Temp := PORT [COMPort [Com] + MSR]; { MODEM STATUS CHANGES }
449     CD [Com] := $80 AND Temp <> 0; { Carrier Detect }
450     CTS [Com] := $10 AND Temp <> 0; { Clear To Send }
451     DSR [Com] := $20 AND Temp <> 0; { Data Set Ready }
452     RI [Com] := $40 AND Temp <> 0; { Ring Indicator }
453 END;
454 2 : BEGIN
455     IF (OutHead [Com] = OutTail [Com]) { TRANSMIT REGISTER EMPTY }
456     THEN
457         PORT [COMPort [Com] + IER] := PORT [COMPort [Com] + IER] AND $FD
458         { If no more data to send, }
459     ELSE { shut off the transmitter. }
460     BEGIN { Otherwise, send the next }
461         PORT [COMPort [Com] + THR] := OutBuffer [Com, OutHead [Com]];
462         { byte, and remove it from }
463         OutHead [Com] := (OutHead [Com] + 1) MOD (MaxOutSize + 1);
464         { the buffer. }
465     END;
466 END;
467 4 : BEGIN
468     IF (InTail [Com] + 1) MOD (MaxInSize + 1) <> InHead [Com]
469     { RECEIVE REGISTER FULL }
470     THEN
471     BEGIN { If the buffer is not full, }
472         InBuffer [Com, InTail [Com]] := PORT [COMPort [Com] + RHR];
473         { add the character and set }
474         InTail [Com] := (InTail [Com] + 1) MOD (MaxInSize + 1);
475         { the queue buffer pointer. }
476     END { Otherwise, the character }
477     ELSE { is read but not stored. }
478     BEGIN
479         IF (PORT [COMPort [Com] + RHR] = $00) THEN { DO Nothing };
480     END;
481 END;
482 6 : BEGIN { LINE STATUS CHANGE & ERROR }
483     Temp := PORT [COMPort [Com] + LSR] AND $1E;
484     { Just count the errors }
485     IF (Temp AND $02 <> 0) THEN INC (Overrun [Com]);
486     { Overrun Error }
487     IF (Temp AND $04 <> 0) THEN INC (Parity [Com]);
488     { Parity Error }
489     IF (Temp AND $08 <> 0) THEN INC (Framing [Com]);
490     { Framing Error }
491     IF (Temp AND $10 <> 0) THEN INC (Break [Com]);
492     { Break Interrupt }
493 END;
494 END;
495 DisableInts; { -Accessing 8259 hardware }

496 PORT [$21] := IMR; { -Enable Comm_TP4 interrupts }
497 PORT [$20] := $20; { -Notify 8259 that interrupt }
498 END; { has been completed. }
499 { $F- }
500
501                                { -Interrupt handlers MUST be }
502 { $F+ } { FAR calls. }
503 PROCEDURE IntHandler2; INTERRUPT;
504
505 Const
506     com = 2;
507
508 Var
509     imr,
510     temp : Byte;
511
512 BEGIN
513     IMR := PORT [$21]; { -Backup IMR for later use }
514     PORT [$21] := IMR OR IRQMask; { -Disable Comm_TP4 interrupts }
515     EnableInts; { -Allow other interrupts }
516
517     PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;
518
519                                { -Set THR, RHR & IER active }
520 CASE PORT [COMPort [Com] + IIR] AND $06 OF { -Identify interrupt type }
521 0 : BEGIN
522     Temp := PORT [COMPort [Com] + MSR]; { MODEM STATUS CHANGES }
523     CD [Com] := $80 AND Temp <> 0; { Carrier Detect }
524     CTS [Com] := $10 AND Temp <> 0; { Clear To Send }
525     DSR [Com] := $20 AND Temp <> 0; { Data Set Ready }
526     RI [Com] := $40 AND Temp <> 0; { Ring Indicator }
527 END;
528 2 : BEGIN
529     IF (OutHead [Com] = OutTail [Com]) { TRANSMIT REGISTER EMPTY }
530     THEN
531         PORT [COMPort [Com] + IER] := PORT [COMPort [Com] + IER] AND $FD
532         { If no more data to send, }
533     ELSE { shut off the transmitter. }
534     BEGIN { Otherwise, send the next }
535         PORT [COMPort [Com] + THR] := OutBuffer [Com, OutHead [Com]];
536         { byte, and remove it from }
537         OutHead [Com] := (OutHead [Com] + 1) MOD (MaxOutSize + 1);
538         { the buffer. }
539     END;
540 END;
541 4 : BEGIN
542     IF (InTail [Com] + 1) MOD (MaxInSize + 1) <> InHead [Com]
543     { RECEIVE REGISTER FULL }
544     THEN
545     BEGIN { If the buffer is not full, }
546         InBuffer [Com, InTail [Com]] := PORT [COMPort [Com] + RHR];
547         { add the character and set }
548         InTail [Com] := (InTail [Com] + 1) MOD (MaxInSize + 1);
549         { the queue buffer pointer. }
550     END { Otherwise, the character }

```

```

551         ELSE                                { is read but not stored. } 606
552         BEGIN                                607
553         IF (PORT [COMPort [Com] + RHR] = $00) THEN { DO Nothing }; 608
554         END;                                609
555     END;                                610
556     6 : BEGIN                                { LINE STATUS CHANGE & ERROR } 611
557         Temp := PORT [COMPort [Com] + LSR] AND $1E; 612
558         { Just count the errors } 613
559         IF (Temp AND $02 <> 0) THEN INC (Overrun [Com]); 614
560         { Overrun Error } 615
561         IF (Temp AND $04 <> 0) THEN INC (Parity [Com]); 616
562         { Parity Error } 617
563         IF (Temp AND $08 <> 0) THEN INC (Framing [Com]); 618
564         { Framing Error } 619
565         IF (Temp AND $10 <> 0) THEN INC (Break [Com]); 620
566         { Break Interrupt } 621
567     END;                                622
568 END;                                623
569 DisableInts;                                {-Accessing 8259 hardware } 624
570 PORT [$21] := IMR;                                {-Enable Comm_TP4 interrupts } 625
571 PORT [$20] := $20;                                {-Notify 8259 that interrupt } 626
572 END;                                { has been completed. } 627
573 {$F-}                                628
574                                629
575                                630
576 {$F+}                                {-Interrupt handlers MUST be } 631
577 PROCEDURE IntHandler3; INTERRUPT;                                { FAR calls. } 632
578                                633
579 Const                                634
580     com = 3;                                635
581                                636
582 Var                                637
583     imr,                                638
584     temp : Byte;                                639
585                                640
586 BEGIN                                641
587     IMR := PORT [$21];                                {-Backup IMR for later use } 642
588     PORT [$21] := IMR OR IRQMask;                                {-Disable Comm_TP4 interrupts } 643
589     EnableInts;                                {-Allow other interrupts } 644
590                                645
591     PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F; 646
592                                647
593     {-Set THR, RHR & IER active } 648
594     CASE PORT [COMPort [Com] + IIR] AND $06 OF {-Identify interrupt type } 649
595     0 : BEGIN                                650
596         Temp := PORT [COMPort [Com] + MSR]; { MODEM STATUS CHANGES } 651
597         CD [Com] := $80 AND Temp <> 0; { Carrier Detect } 652
598         CTS [Com] := $10 AND Temp <> 0; { Clear To Send } 653
599         DSR [Com] := $20 AND Temp <> 0; { Data Set Ready } 654
600         RI [Com] := $40 AND Temp <> 0; { Ring Indicator } 655
601     END;                                656
602     2 : BEGIN                                657
603         IF (OutHead [Com] = OutTail [Com]) { TRANSMIT REGISTER EMPTY } 658
604         THEN                                659
605         PORT [COMPort [Com] + IER] := PORT [COMPort [Com] + IER] AND $FD 660

```

```

                                { If no more data to send, }
                                { shut off the transmitter. }
                                { Otherwise, send the next }
                                PORT [COMPort [Com] + THR] := OutBuffer [Com, OutHead [Com]];
                                { byte, and remove it from }
                                OutHead [Com] := (OutHead [Com] + 1) MOD (MaxOutSize + 1);
                                { the buffer. }
                                END;
                                4 : BEGIN
                                IF (InTail [Com] + 1) MOD (MaxInSize + 1) <> InHead [Com]
                                    { RECEIVE REGISTER FULL }
                                THEN
                                    BEGIN
                                        { If the buffer is not full, }
                                        InBuffer [Com, InTail [Com]] := PORT [COMPort [Com] + RHR];
                                        { add the character and set }
                                        InTail [Com] := (InTail [Com] + 1) MOD (MaxInSize + 1);
                                        { the queue buffer pointer. }
                                    END
                                ELSE
                                    { Otherwise, the character }
                                    { is read but not stored. }
                                    BEGIN
                                        IF (PORT [COMPort [Com] + RHR] = $00) THEN { DO Nothing };
                                    END;
                                END;
                                6 : BEGIN                                { LINE STATUS CHANGE & ERROR }
                                Temp := PORT [COMPort [Com] + LSR] AND $1E;
                                { Just count the errors }
                                IF (Temp AND $02 <> 0) THEN INC (Overrun [Com]);
                                { Overrun Error }
                                IF (Temp AND $04 <> 0) THEN INC (Parity [Com]);
                                { Parity Error }
                                IF (Temp AND $08 <> 0) THEN INC (Framing [Com]);
                                { Framing Error }
                                IF (Temp AND $10 <> 0) THEN INC (Break [Com]);
                                { Break Interrupt }
                                END;
                                END;
                                DisableInts;                                {-Accessing 8259 hardware }
                                PORT [$21] := IMR;                                {-Enable Comm_TP4 interrupts }
                                PORT [$20] := $20;                                {-Notify 8259 that interrupt }
                                END;                                { has been completed. }
                                {$F-}
                                650 {$F+}                                {-Interrupt handlers MUST be }
                                651 PROCEDURE IntHandler4; INTERRUPT;                                { FAR calls. }
                                652 Const
                                653     com = 4;
                                654 Var
                                655     imr,
                                656     temp : Byte;
                                657 BEGIN

```

```

661 IMR := PORT [$21];           {-Backup IMR for later use }
662 PORT [$21] := IMR OR IRQMask; {-Disable Comm TP4 interrupts}
663 EnableInts;                   {-Allow other interrupts }
664
665 PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;
666
667                               {-Set THR, RHR & IER active }
668 CASE PORT [COMPort [Com] + IIR] AND $06 OF {-Identify interrupt type }
669 0 : BEGIN
670     Temp := PORT [COMPort [Com] + MSR]; { MODEM STATUS CHANGES }
671     CD [Com] := $80 AND Temp <> 0; { Carrier Detect }
672     CTS [Com] := $10 AND Temp <> 0; { Clear To Send }
673     DSR [Com] := $20 AND Temp <> 0; { Data Set Ready }
674     RI [Com] := $40 AND Temp <> 0; { Ring Indicator }
675 END;
676 2 : BEGIN
677     IF (OutHead [Com] = OutTail [Com]) { TRANSMIT REGISTER EMPTY }
678     THEN
679         PORT [COMPort [Com] + IER] := PORT [COMPort [Com] + IER] AND $FD
680         { If no more data to send, }
681         { shut off the transmitter. }
682     ELSE
683         BEGIN
684             PORT [COMPort [Com] + THR] := OutBuffer [Com, OutHead [Com]];
685             { byte, and remove it from }
686             OutHead [Com] := (OutHead [Com] + 1) MOD (MaxOutSize + 1);
687             { the buffer. }
688         END;
689     4 : BEGIN
690         IF (InTail [Com] + 1) MOD (MaxInSize + 1) <> InHead [Com]
691         { RECEIVE REGISTER FULL }
692         THEN
693             BEGIN
694                 { If the buffer is not full, }
695                 InBuffer [Com, InTail [Com]] := PORT [COMPort [Com] + RHR];
696                 { add the character and set }
697                 InTail [Com] := (InTail [Com] + 1) MOD (MaxInSize + 1);
698                 { the queue buffer pointer. }
699                 { Otherwise, the character }
700             END
701             ELSE
702                 { is read but not stored. }
703                 BEGIN
704                     IF (PORT [COMPort [Com] + RHR] = $00) THEN { DO Nothing };
705                 END;
706         6 : BEGIN
707             { LINE STATUS CHANGE & ERROR }
708             Temp := PORT [COMPort [Com] + LSR] AND $1E;
709             { Just count the errors }
710             IF (Temp AND $02 <> 0) THEN INC (Overrun [Com]);
711             { Overrun Error }
712             IF (Temp AND $04 <> 0) THEN INC (Parity [Com]);
713             { Parity Error }
714             IF (Temp AND $08 <> 0) THEN INC (Framing [Com]);
715             { Framing Error }
716             IF (Temp AND $10 <> 0) THEN INC (Break [Com]);
717             { Break Interrupt }
718         END;
719     END;
720
721 END;
722
723 { This procedure installs and enables the specified serial port interrupt. All
724 port input line monitoring variables are set to match the actual line
725 states. The old serial port interrupt vector is saved so it can be
726 reinstalled when we remove our serial port interrupt. DTR and RTS are
727 forced to the ready state. The 8250 interrupts are enabled by ORing the
728 MCR with $08. To enable all four 8250 interrupt types, write $0F to the
729 IER. (Receive Buffer Full, Line Status, & Modem Status interrupts are
730 enabled by writing $0D to the IER). ORing $EF with PORT [$21] enables
731 IRQ4 (COM1 or COM3), while $F7 enables IRQ3 (COM2 or COM4). Hardware
732 interrupts should be disabled during the installation process since the
733 8259 ports are being set. Error checking is always in place since a crash
734 could occur if it is used when a the handler for the same port has already
735 been installed. }
736
737 PROCEDURE InstallInt (Com : BYTE);
738 VAR
739     Temporary : BYTE;
740 BEGIN
741     IF ValidPort (Com, 0, 'InstallInt') {-Error checking important! }
742     THEN
743         BEGIN
744             DisableInts; {-Accessing 8259 hardware }
745             Temporary := PORT [COMPort [Com] + MSR];
746             CD [Com] := ($80 AND Temporary <> 0); {-Carrier Detect status }
747             CTS [Com] := ($10 AND Temporary <> 0); {-Clear to Send status }
748             DSR [Com] := ($20 AND Temporary <> 0); {-Data Set Ready status }
749             RI [Com] := ($40 AND Temporary <> 0); {-Ring Indicator status }
750             Temporary := PORT [COMPort [Com] + LSR]; {-Reset interrupts that were }
751             Temporary := PORT [COMPort [Com] + RHR]; {- waiting to be processed. }
752             Temporary := 1 SHL IRQNmbr [Com]; {-If other port using same }
753             IF (IRQMask AND Temporary) = 0 { IRQ then nothing must be }
754             THEN BEGIN { done to 8259 or vectors. }
755                 IRQMask := IRQMask OR Temporary; {-Update interrupt record }
756                 GETINTVEC ($08 + IRQNmbr [Com], OldIntVector [Com]);
757                 {-Save old interrupt vector }
758             END;
759             Case Com Of
760             1: SETINTVEC ($08 + IRQNmbr [Com], @IntHandler1);
761             2: SETINTVEC ($08 + IRQNmbr [Com], @IntHandler2);
762             3: SETINTVEC ($08 + IRQNmbr [Com], @IntHandler3);
763             4: SETINTVEC ($08 + IRQNmbr [Com], @IntHandler4);
764             End;
765             {-Install Comm TP4 vector }
766             PORT [$21] := PORT [$21] AND NOT Temporary;
767             {-Enable 8259 IRQ handling }
768         END;
769     PORT [COMPort [Com] + MCR] := PORT [COMPort [Com] + MCR] OR $08;
770     {-Enable 8250 interrupt line }
771     PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;

```

```

771      PORT [COMPort [Com] + IER] := $01;      {-Set THR/RHR/IER active }
772      IntInstalled [Com] := TRUE;              {-Enable 8250 interrupts }
773      Set_DTR_RTS (Com, TRUE);                {-The interrupt is installed }
774      EnableInts;                             {-DTR/RTS on so the other }
775      END;                                    { device knows we are ready }
776      END;                                    { to receive data. }
777 END;
778
779 { This procedure removes the specified serial port interrupt and reinstalls
780 the original interrupt vectors. DTR & RTS are set OFF and 8250 interrupt
781 line is disabled by ANDing the MCR with $F7. All 8250 interrupt types are
782 disabled by writing $00 to the IER. Oring $10 with PORT [$21] disables
783 IRQ4 (COM1 or COM3), while $08 disables IRQ3 (COM2 or COM4). Hardware
784 interrupts must be disabled since the 8259 ports are being set. Some error
785 checking is always in place since attempting RemoveInt on a handler which
786 has not been installed can cause the computer to eventually crash.}
787
788 PROCEDURE RemoveInt (Com : BYTE);
789 VAR
790     Temporary : BYTE;
791 BEGIN
792     IF ValidPort (Com, 1, 'RemoveInt')      {-Error checking important! }
793     THEN
794         BEGIN
795             DisableInts;                    {-Accessing 8259 hardware }
796             Set_DTR_RTS (Com, FALSE);        {-DTR/RTS off }
797             IntInstalled [Com] := FALSE;      {-Uninstalling interrupt }
798             PORT [COMPort [Com] + MCR] := PORT [COMPort [Com] + MCR] AND $F7;
799             {-Disable 8250 interrupt line}
800             PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;
801             {-Set THR, THE & IER active }
802             PORT [COMPort [Com] + IER] := $00; {-Disable 8250 interrupts }
803             Temporary := 1 SHL IRQNmbR [Com]; {-Get bit for IRQ mask }
804             IF (IRQMask AND Temporary) <> 0   {-If no other Comm_TP4 }
805             THEN BEGIN                       { interrupt is on this IRQ: }
806                 PORT [$21] := PORT [$21] OR Temporary;
807                 { Disable 8259 IRQ handling, }
808                 SETINTVEC ($08 + IRQNmbR [Com], OldIntVector [Com]);
809                 { Replace original vector, }
810                 IRQMask := IRQMask AND NOT Temporary;
811                 { and update IRQ mask. }
812             END;
813             EnableInts;                      {-Done with 8259 setup }
814         END;
815     END;
816
817 { This procedure is provided for situations where you want to be sure that
818 any of the serial port buffers are empty. This could be used for aborting
819 an interrupt driven transmission or for clearing unwanted data from the
820 input buffers. The first parameter is the buffer number and the second
821 parameter is TRUE for the input buffer or FALSE for the output buffer.
822 A buffer number of 0 causes all buffers to be emptied. }
823
824 PROCEDURE EmptyBuffer (Buffer : BYTE; TrueInFalseOut : BOOLEAN);
825 VAR
826     LoopVar : BYTE;
827 BEGIN
828     FOR LoopVar := 1 to MaxPorts DO
829         IF (Buffer = 0) OR (LoopVar = Buffer)
830         THEN BEGIN
831             DisableInts;                    {-Disable buffer activity }
832             IF TrueInFalseOut
833             THEN InHead [LoopVar] := InTail [LoopVar]
834             ELSE OutHead [LoopVar] := OutTail [LoopVar];
835             {-Clear an input buffer }
836             {-Clear an output buffer }
837             EnableInts;                      {-Reenable buffer activity }
838         END;
839     END;
840
841 { This function/procedure writes character or string data to the serial port.
842 It does this by reading and writing to the 8250 communications chip. This
843 is an example that may be modified to suit your purposes. As is, it pauses
844 the program while it sends the data. If it cannot send a character after
845 65535 tries, it aborts the sending process. Use the conditional
846 compilation directive $DEFINE FWriteCOM in order to make WriteCOM a
847 function which returns a BOOLEAN TRUE if the transmission did not timeout.
848 The statement: (PORT [COMPort [Com]+LSR] AND $20) <> $20 indicates when
849 the THR is ready for a new character to send. CTS and DSR are not checked,
850 but if you want to check for them (PORT [COMPort [Com]+MSR] AND $30) must
851 equal $30.}
852
853 {$IFDEF FWriteCOM}                        {-A $DEFINE/$UNDEF FWriteCOM }
854 FUNCTION WriteCOM (Com : BYTE; Data : STRING) : BOOLEAN;
855 {$ELSE}                                  { directive determines if }
856 PROCEDURE WriteCOM (Com : BYTE; Data : STRING); { a procedure. }
857 {$ENDIF}
858
859 { WriteCOM is a function or }
860 VAR
861     LoopVar,                                {-Pointer to output char }
862     TimeLoop : WORD;                        {-Timeout counter variable }
863     Timeout, Ready : BOOLEAN;               {-TRUE if port timed out }
864 BEGIN
865     {$IFDEF ErrorChecking}
866     IF NOT ValidPort (Com, 1, 'WriteCOM') THEN EXIT;
867     {-Optional error trapping }
868     {$ENDIF}
869     LoopVar := 0;
870     Timeout := FALSE;
871     WHILE (LoopVar < LENGTH (Data)) AND NOT Timeout DO
872         BEGIN
873             INC (LoopVar);
874             TimeLoop := 0;
875             REPEAT
876                 Ready := (PORT [COMPort [Com]+LSR] AND $20) <> 0;
877                 INC (TimeLoop);
878                 UNTIL Ready OR (TimeLoop = 65535);
879             IF Ready

```



```

881     THEN BEGIN
882         PORT [COMPort [Com]+LCR] := PORT [COMPort [Com]+LCR] AND $7F;
883         { Allow THR, RHR & IER access }
884         PORT [COMPort [Com]+THR] := ORD (Data [LoopVar]);
885         { Put the data to send in }
886         { the THR. }
887     ELSE BEGIN
888         TimeOut := TRUE; { WriteCOM aborts if the THR }
889         { $IFDEF ErrorChecking } { takes too long to become }
890         MakeError (4, Com, 'WriteCOM'); { empty & optionally creates }
891         { $ENDIF } { an error condition. }
892     END;
893 END;
894 { $IFDEF FWriteCOM } { With a compiler directive }
895 WriteCOM := NOT TimeOut; { of $DEFINE FWriteCOM, then }
896 { $ENDIF } { WriteCOM is a function and }
897 END; { returns TRUE if no timeout }
898
899 { This procedure is an example of how to write an interrupt driven send
900 routine. The main idea is that you add data to the output buffer, then
901 you get things going by manually placing one byte into the transmitter
902 holding register. After doing this, the rest of the buffer will be sent
903 automatically. Strenuous testing of this procedure under very high data
904 rates has not been done, and it might be possible to rewrite it to provide
905 better throughput. Data is the information to send to the port in either
906 CHAR or STRING form. It is impractical to use this procedure for sending
907 single characters since it calls WriteCOM at least once. It is best suited
908 for high volume data transfers. Interrupts should be off during buffer
909 operations. }
910
911 PROCEDURE IWriteCOM (Com : BYTE; Data : STRING);
912 VAR
913     BuffFull : BOOLEAN; { TRUE if output buffer full }
914     StartChr : CHAR; { Temporary Buffer }
915     Loop : WORD; { Points to current Data item }
916 BEGIN
917     { $IFDEF ErrorChecking }
918     IF NOT ValidPort (Com, 1, 'IWriteCOM') THEN EXIT;
919     { $ENDIF }
920     Loop := 1;
921     PORT [COMPort [Com]+LCR] := PORT [COMPort [Com]+LCR] AND $7F;
922     { Enable access to 8250 IER }
923     { Load the output buffer one }
924     WHILE (Loop <= LENGTH (Data)) DO { byte at a time. }
925     BEGIN { During buffer operations }
926         DisableInts;
927         BuffFull := (OutTail [Com] + 1) MOD (MaxOutSize+1) = OutHead[Com];
928         { Is the buffer full? }
929         IF NOT BuffFull { If not, add a character to }
930         THEN BEGIN { buffer and update pointers }
931             OutBuffer [Com, OutTail [Com]] := ORD (Data [Loop]);
932             OutTail [Com] := (OutTail [Com] + 1) MOD (MaxOutSize + 1);
933             { NOTE: Interrupts should be }
934             INC (Loop); { enabled within the Loop so }
935             { the interrupt can empty }

```

```

936     EnableInts; { -the buffer as we fill it. }
937     IF BuffFull OR (Loop > LENGTH (Data)) { -Check the interrupt status }
938     THEN { if the buffer gets full or }
939     BEGIN { after data is all loaded. }
940         IF (PORT [COMPort [Com] + IER] AND $02 <> 2)
941         THEN { -If the transmit interrupt }
942             { is not on, start it up. }
943         BEGIN
944             DisableInts;
945             StartChr := CHR (OutBuffer [Com, OutHead [Com]]);
946             { Get the first character & }
947             OutHead [Com] := (OutHead [Com] + 1) MOD (MaxOutSize + 1);
948             { take it out of the buffer. }
949             PORT [COMPort [Com]+IER] := PORT [COMPort [Com]+IER] OR $02;
950             { -Turn transmit interrupt on }
951             EnableInts;
952             { $IFDEF FWriteCOM } { -Kickstart the transmitter }
953             IF WriteCOM (Com, StartChr) THEN {}; { interrupt by sending one }
954             { $ELSE } { character. WriteCOM is }
955             WriteCOM (Com, StartChr); { used for simplicity. }
956             { $ENDIF }
957         END;
958     END;
959 END;
960
961 END;
962
963
964 { This function is an example of how to get a character from the serial port.
965 As is, if the buffer is empty, it waits until a character arrives, so this
966 will not work for the TTY emulation. The interrupts are always disabled
967 when the buffer pointers are checked or modified. Beware! Do not
968 completely disable interrupts in the wait loop or else you never will get
969 a character if there is not one there already. }
970
971 FUNCTION ReadCOM (Com : BYTE) : CHAR;
972 VAR
973     CharReady : BOOLEAN; { -TRUE if there is data in }
974     { the input buffer }
975 BEGIN
976     { $IFDEF ErrorChecking }
977     IF NOT ValidPort (Com, 1, 'ReadCOM') THEN EXIT;
978     { $ENDIF }
979     CharReady := FALSE;
980     REPEAT { -Wait for data to arrive }
981     BEGIN
982         DisableInts;
983         CharReady := InTail [Com] <> InHead [Com]; { -Check to see if buffer is }
984         EnableInts; { empty }
985     UNTIL CharReady;
986     DisableInts;
987     ReadCOM := CHR(InBuffer [Com, InHead [Com]]); { -Read a character of data }
988     InHead [Com] := (InHead [Com] + 1) MOD (MaxInSize + 1);
989     { -Update the buffer pointer }
990     EnableInts;
991 END;

```

```

991
992 { This function is an example of how to get a character from the serial port.
993 Unlike ReadCOM, this routine returns if no data appears in the buffer for
994 a short period of time. This makes it useful for applications which process
995 data only if it is available. The interrupts are always disabled when the
996 buffer pointers are checked or modified. Beware! Do not completely
997 disable interrupts in the wait loop or else you never will get a character
998 if there is not one there already. Returns TRUE if valid data has been
999 returned. You may optimize the time out period to a shorter one for your
1000 applications. 65535 is the maximum wait time.}
1001
1002
1003 FUNCTION TimedReadCOM (Com : BYTE; VAR Data : CHAR) : BOOLEAN;
1004 VAR
1005   CharReady : BOOLEAN;           {-TRUE if there is data in  }
1006   TimeOut : WORD;               { the input buffer      }
1007 BEGIN
1008   {$IFDEF ErrorChecking}
1009   IF NOT ValidPort (Com, 1, 'TimedReadCOM') THEN EXIT;
1010   {-Optional error trapping  }
1011   {$ENDIF}
1012   TimeOut := 0;
1013   REPEAT
1014     DisableInts;
1015     CharReady := InTail [Com] <> InHead [Com]; {-Is the buffer empty or not }
1016     EnableInts;
1017     INC (TimeOut);                       {-Increment the timer  }
1018   UNTIL CharReady OR (TimeOut = 65535);  {-Set the maximum time to  }
1019   IF CharReady
1020     THEN BEGIN                          { wait for data here. Lower }
1021       DisableInts;
1022       Data := CHR (InBuffer [Com, InHead [Com]]);
1023       {-If data became available,  }
1024       InHead [Com] := (InHead [Com] + 1) MOD (MaxInSize + 1);
1025       { read a character, and set  }
1026       EnableInts;
1027       { the buffer pointers.      }
1028     ELSE MakeError (5, Com, 'TimedReadCOM'); {-Record the timeout error  }
1029   TimedReadCOM := CharReady;
1030 END;
1031
1032 {$F+}                                {-VERY IMPORTANT! When the  }
1033 PROCEDURE RemoveIntOnExit;           { program quits normally or }
1034 BEGIN                                { abnormally, the interrupt }
1035   FOR Loop := 1 TO MaxPorts DO      { handler is automatically }
1036     IF IntInstalled [Loop]           { uninstalled when Turbo   }
1037     THEN RemoveInt (Loop);           { invokes this procedure.  }
1038   ExitProc := ExitSave;              {-Return control to the    }
1039 END;                                  { original exit procedure  }
1040 {$F-}
1041
1042 Function Getcharbuff( comport: Byte): Char;
1043 {*****}
1044 { Function to retrieve one character from the inputbuffer. #00 is returned}
1045 { whenever, there was no character in the inputbuffer}
1046 {
1047   Input : comportnumber}
1048 {
1049   Output: character}
1050 {*****}
1051 Var
1052   dataready : Boolean;
1053   tempcharbuff : Char;
1054 BEGIN
1055   If restoreflag[comport]
1056   Then Begin
1057     tempCharbuff:= restorebuff[comport,1];
1058     If (Length( restorebuff[comport]) > 1)
1059     Then restorebuff[comport]:= Copy( restorebuff[comport], 2,
1060                                     Length( restorebuff[comport])-1)
1061     Else restoreflag[comport]:= False;
1062   End
1063   Else Begin
1064     Disableints;
1065     dataready:= ( InTail[ comport] <> Inhead[ comport]);
1066     Enableints;
1067     If dataready
1068     Then Begin
1069       Disableints;
1070       tempcharbuff:= Char( inbuffer[ comport,
1071                           Inhead[ comport]]);
1072       Inhead[ comport]:= ( Inhead[ comport] + 1) MOD
1073                           ( Maxinsize+1);
1074       Enableints;
1075     End
1076     Else tempcharbuff:=#00;
1077   End;
1078   GetCharbuff:= tempcharbuff;
1079 End;
1080
1081
1082 Function Lookbuff( comport: Byte): Char;
1083 {*****}
1084 { Function to see the next character in the inputbuffer. The character is }
1085 { not retrieved. #00 is returned whenever, there was no character in the }
1086 { inputbuffer.
1087   Input : comportnumber}
1088 {
1089   Output: character}
1090 {*****}
1091 Var
1092   dataready : Boolean;
1093 BEGIN
1094   If restoreflag[comport]
1095   Then Lookbuff:= restorebuff[comport,1]
1096   Else Begin
1097     Disableints;
1098     dataready:= ( InTail[ comport] <> Inhead[ comport]);
1099     Enableints;
1100     If dataready

```

```

1101         Then Begin
1102             Disableints;
1103             Lookbuff:= Char( inbuffer[ comport, Inhead[ comport]]);
1104             Enableints;
1105         End
1106     Else Lookbuff:=#00;
1107 End;
1108 End;
1109
1110
1111 Function Charsinbuff( comport: Byte): Integer;
1112 {*****}
1113 { Function which returns the number of characters in the inputbuffer for a }
1114 { specific comport.      Input : comportnumber }
1115 {      Output: number of characters in the buffer }
1116 {*****}
1117
1118 Var
1119     temp      : Integer;
1120 Begin
1121     Disableints;
1122     If ( intail[ comport] >= inhead[ comport])
1123     Then temp:= intail[ comport] - inhead[ comport]
1124     Else temp:= intail[ comport] + ( maxinsize - inhead[ comport]);
1125     If Restoreflag[comport]
1126     Then temp:= temp + Length( restorebuff[ comport]);
1127
1128     charsinbuff:= temp;
1129     Enableints;
1130 End;
1131
1132
1133 Procedure Restore_buffer( comport: Byte; line: String);
1134
1135 Var
1136     x      : Word;
1137
1138 Begin
1139     Disableints;
1140     restorebuff[comport]:= line;
1141     restoreflag[comport]:= True;
1142     Enableints;
1143 End;
1144
1145 { The following code is executed when any program which uses this unit first
1146   Changeable defaults are starts up.  It performs all necessary
1147   initializations. marked with a '*' }
1148
1149 BEGIN
1150     ExitSave := ExitProc;          { -VERY IMPORTANT! This lets }
1151     ExitProc := @RemoveIntOnExit;  { the program halt safely.  }
1152     FOR Loop := 1 TO MaxPorts DO
1153         BEGIN
1154             InHead [Loop] := 0;      { -Default all buffers set to }
1155             InTail [Loop] := 0;      { empty on startup.          }

```

```

1156     OutHead [Loop] := 0;
1157     OutTail [Loop] := 0;
1158     DTR_RTS [Loop] := FALSE;        { *Default DTR/RTS setting on }
1159     IntInstalled [Loop] := FALSE;   { -Default interrupts not on }
1160     CD [Loop] := ($80 AND PORT [COMPort [Loop]+MSR] <> 0);
1161                                     { -Carrier Detect status set }
1162     CTS [Loop] := ($10 AND PORT [COMPort [Loop]+MSR] <> 0);
1163                                     { -Clear To Send status set }
1164     DSR [Loop] := ($20 AND PORT [COMPort [Loop]+MSR] <> 0);
1165                                     { -Data Set Ready status set }
1166     RI [Loop] := ($40 AND PORT [COMPort [Loop]+MSR] <> 0);
1167                                     { -Ring Indicator status set }
1168     Framing [Loop] := 0;            { -Reset framing error count }
1169     Overrun [Loop] := 0;            { -Reset overrun error count }
1170     Parity [Loop] := 0;             { -Reset parity error count }
1171     Break [Loop] := 0;              { -Reset break interrupt count }
1172 END;
1173 ErrorCode := 0;                    { -Default of no port errors }
1174 ErrorPort := 0;
1175 IRQMask := 0;                      { -No interrupts installed }
1176 { $IFDEF NoMessageCode }
1177     ShowMessages := TRUE;           { *Default error messages on }
1178     ErrMsgX := 0;                   { *Default of error messages }
1179     ErrMsgY := 0;                   { *placed at cursor position. }
1180 { $ENDIF }
1181     For cnter:= 1 To Maxports Do
1182         restoreflag[cnter]:= False;
1183 END.
1184

```

## **Appendix E Listings for MIASNAV**

This appendix lists only the files of 'MIASNAV' that are different from 'MIASLOGO'.

```
1 Program MIASSystem;
2
3 {$M 40000, 0, 650000}
4
5 Uses MIASglob, MIAS;
6
7 Var
8   alldata      :   alldatatype;
9   command      :   commandtype;
10  position,
11  filtposition,
12  predposition :   positiontype;
13
14 Begin
15   Init( alldata, position);
16   Repeat
17     SetTimerToGPSIfNotSet;
18     GetUserCommands( command);
19     ExecCommands( command, alldata);
20     GetData( alldata);
21     DispFlags( alldata, position);
22     CalcPos( alldata, position);
23     FilterPosition( position, filtposition);
24     PredictPosition( position, predposition);
25     SendPosition( position);
26   Until Stopcommand( command);
27   CloseDown( alldata, position);
28 End.
```

```

1 Unit MLsglob;
2 {*****}
3 { This unit contains global MLStypes and MLS variables}
4 { See also the ICAO Annex 10 Part 1 page 146 and 147.}
5 {*****}
6
7 Interface
8
9 Uses Miasglob, Miscell;
10
11 Const
12     Valid_Bas1      :      timetype      =      (year :0;
13                                                    month:0;
14                                                    day  :0;
15                                                    hour :0;
16                                                    minute:0;
17                                                    sec  :1;
18                                                    sec100:0);
19     Valid_Bas2      :      timetype      =      (year :0;
20                                                    month:0;
21                                                    day  :0;
22                                                    hour :0;
23                                                    minute:0;
24                                                    sec  :0;
25                                                    sec100:16);
26     Valid_Bas3      :      timetype      =      (year :0;
27                                                    month:0;
28                                                    day  :0;
29                                                    hour :0;
30                                                    minute:0;
31                                                    sec  :1;
32                                                    sec100:0);
33     Valid_Bas4      :      timetype      =      (year :0;
34                                                    month:0;
35                                                    day  :0;
36                                                    hour :0;
37                                                    minute:0;
38                                                    sec  :1;
39                                                    sec100:0);
40     Valid_Bas5      :      timetype      =      (year :0;
41                                                    month:0;
42                                                    day  :0;
43                                                    hour :0;
44                                                    minute:0;
45                                                    sec  :1;
46                                                    sec100:0);
47     Valid_Bas6      :      timetype      =      (year :0;
48                                                    month:0;
49                                                    day  :0;
50                                                    hour :0;
51                                                    minute:0;
52                                                    sec  :1;
53                                                    sec100:0);
54     Valid_AuxA1      :      timetype      =      (year :0;
55                                                    month:0;
56                                                    day  :0;
57                                                    hour :0;
58                                                    minute:0;
59                                                    sec  :1;
60                                                    sec100:0);
61     Valid_AuxA2      :      timetype      =      (year :0;
62                                                    month:0;
63                                                    day  :0;
64                                                    hour :0;
65                                                    minute:0;
66                                                    sec  :1;
67                                                    sec100:0);
68     Valid_AuxA3      :      timetype      =      (year :0;
69                                                    month:0;
70                                                    day  :0;
71                                                    hour :0;
72                                                    minute:0;
73                                                    sec  :1;
74                                                    sec100:0);
75     Valid_AuxA4      :      timetype      =      (year :0;
76                                                    month:0;
77                                                    day  :0;
78                                                    hour :0;
79                                                    minute:0;
80                                                    sec  :1;
81                                                    sec100:0);
82     Valid_AuxB       :      timetype      =      (year :0;
83                                                    month:0;
84                                                    day  :0;
85                                                    hour :0;
86                                                    minute:2;
87                                                    sec  :0;
88                                                    sec100:0);
89     Valid_AuxC       :      timetype      =      (year :0;
90                                                    month:0;
91                                                    day  :0;
92                                                    hour :0;
93                                                    minute:0;
94                                                    sec  :3;
95                                                    sec100:0);
96     Az2thresdist= 3700;
97     AzPropCovNegLim=54;
98     AzPropCovPosLim=54;
99     Cleartype=1;
100
101     MinGP =3;
102     BAZstat=0;
103     DMEstat=1;
104     Azstat=1;
105     Elstat=1;
106
107     AzBW=2;
108     ElBW=2;
109     DMEdist=3387.5;
110

```

```

111 AzMagOr=186;
112 BazMagOr=0;
113
114 BazPropCovNegLim=0;
115 BazPropCovPosLim=0;
116 BazBW=0;
117 {BazStat=0;}{ is already declared}
118
119 MLSident='MSC';
120
121 AzOff =0;
122 Az2MLSdatdist =3394;
123 AzAlignRun=0;
124 AzCoorSyst=0;
125
126 Eloff=-80;
127 MLSdat2thres =275;
128 ElHeight =1.2;
129 {MLS datum point elevation = -3}
130 {Runway threshold height = 0}
131
132 DMEoff=56;
133 DME2MLSdatDist=3385;
134 {DME antenna height = 5}
135 {Runway stopend distance = 3122}
136
137 BAZoff=0;
138 BAZ2MLSdatDist=0;
139 BAZAlignRun=0;
140
141 Type
142 Basic1type = Record
143     Az2thresDist : Integer;
144     { Azimuth to threshold distance}
145     AzPropCovNegLim,
146     { Azimuth proportional coverage,
147     negative limit}
148     AzPropCovPosLim,
149     { idem, positive limit}
150     ClearType : Byte;
151     { clearance signal type}
152 End;
153 Basic2type = Record
154     MinGP : Real;
155     { minimum glide path}
156     BAZstat, { Back Azimuth status}
157     DMEstat, { DME status}
158     Azstat, { Azimuth status}
159     Elstat : Byte;
160     { Elevation status}
161 End;
162 Basic3type = Record
163     AzBW, { Azimuth beamwidth}
164     ElBW, { Elevation beamwidth}
165     DMEdist : Real;
166
167
168 Basic4type = Record
169     AzMagOr, { Azimuth magnetic orientation}
170     BazMagOr : Integer;
171     { BackAzimuth magnetic orientation}
172 End;
173 Basic5type = Record
174     BazPropCovNegLim,
175     { Back Azimuth Proportional
176     Coverage Negative limit}
177     BazPropCovPosLim: Byte;
178     { idem positive limit}
179     BazBW : Real;
180     { Back Azimuth beamwidth}
181     BazStat : Byte;
182     { Back Azimuth status}
183 End;
184 Basic6type = Record
185     MLSident: String[3];
186 End;
187
188
189 AuxA1type = Record
190     AzOff, { Azimuth Antenna offset}
191     Az2MLSdatdist: Integer;
192     { Azimuth antenna to MLS datum
193     point distance}
194     AzAlignRun : Real;
195     { Azimuth Alignment with Runway
196     centreline}
197     AzCoorSyst : Byte;
198     { Azimuth Antenna Coordinate
199     system}
200 End;
201
202 AuxA2type = Record
203     Eloff, { Elevation antenna offset}
204     MLSdat2thres: Integer;
205     { MLS datum point 2 threshold
206     distance}
207     ElHeight : Real;
208     { Elevation Antenna Height}
209 End;
210
211 AuxA3type = Record
212     DMEoff, { DME offset}
213     DME2MLSdatDist: Integer;
214     { DME to MLS datum point distance}
215 End;
216
217 AuxA4type = Record
218     BAZoff, { Back azimuth antenna offset}
219     BAZ2MLSdatDist: Integer;
220     { Back azimuth to MLS datum
221     point distance}
222     BAZAlignRun: Real;
223     { Back azimuth alignment with

```

```

221                               runway centre line)
222                               End;
223 Discretestyp= Record
224     antenna : Byte;
225     test : Byte;
226     Azsource : Byte;
227     Azselwarn : Byte;
228     Bazselwarn : Byte;
229     GPselwarn : Byte;
230     BAZavail : Byte;
231     BAZdeven : Byte;
232     Tuningcom : Byte;
233     nr1antssel : Byte;
234     changeinh : Byte;
235     tunPrtsel : Byte;
236                               End;
237
238 MLSinttype = Record
239     Bas1 : Basic1type;
240     Bas2 : Basic2type;
241     Bas3 : Basic3type;
242     Bas4 : Basic4type;
243     Bas5 : Basic5type;
244     Bas6 : Basic6type;
245     AuxA1 : AuxA1type;
246     AuxA2 : AuxA2type;
247     AuxA3 : AuxA3type;
248     AuxA4 : AuxA4type;
249     AuxB,
250     AuxC : ADWtype;
251
252     Bas1_Time : timetype;
253     Bas2_Time : timetype;
254     Bas3_Time : timetype;
255     Bas4_Time : timetype;
256     Bas5_Time : timetype;
257     Bas6_Time : timetype;
258     AuxA1_Time : timetype;
259     AuxA2_Time : timetype;
260     AuxA3_Time : timetype;
261     AuxA4_Time : timetype;
262     AuxB_time,
263     AuxC_time : timetype;
264
265     DMErange,
266     ELangle,
267     AZangle,
268     BAZangle : Double;
269     Discretes : Discretestyp;
270     Leftclr,
271     Rightclr : Boolean;
272     ElantInUse,
273     AzantInUse : Byte;
274
275     Bas1_flag,

```

```

276 Bas2_flag,
277 Bas3_flag,
278 Bas4_flag,
279 Bas5_flag,
280 Bas6_flag,
281 Auxa1_flag,
282 Auxa2_flag,
283 Auxa3_flag,
284 Auxa4_flag,
285 AuxB_flag,
286 AuxC_flag,
287
288 ELangle_flag,
289 AZangle_flag,
290 BAZangle_flag,
291 DME_flag,
292 discretes_flag: Boolean;
293
294 flag : Boolean;

```

End;

```

297
298 Implementation
299
300 Begin
301 End.

```



```

1 Unit MIAS;
2
3 Interface
4
5 {$N+,E+}
6
7
8 Uses MIASglob;
9
10
11 Procedure Init( Var alldata: alldatatype; Var position: positiontype);
12 {*****}
13 {Initialise the MIAS system. Open a setupfile and read some values.
14   Input :-
15   Output:status; whether or not a device present}
16 {*****}
17
18
19 Procedure DispFlags( alldata: alldatatype; position: positiontype);
20 {*****}
21 {Display the flags on the screen or in a file. The flags represent the valid-
22 ness of a certain device. The procedure uses only the flag-fields of the
23 peripheral-fields in alldata.
24   Input :alldata
25   Output:flags on screen or file}
26 {*****}
27
28
29 Procedure GetUserCommands( Var command: commandtype);
30 {*****}
31 {Retrieve commands for the peripherals from the keyboard.
32   Input :-
33   Output:commands string in upcase}
34 {*****}
35
36
37 Procedure ExecCommands( command: commandtype; alldata: alldatatype);
38 {*****}
39 {Send commands to peripherals, or execute the commands on the host.
40   Input :commands
41   Output:-}
42 {*****}
43
44
45 Procedure GetData( Var alldata: alldatatype);
46 {*****}
47 {Retrieve information from the peripherals.
48   Input :-
49   Output:alldata; information from peripherals}
50 {*****}
51
52
53 Procedure CalcPos( Var alldata: alldatatype; Var position: positiontype);
54 {*****}
55 {Calculate the position from the information available.
56
57   Input :alldata
58   Output:position}
59 {*****}
60
61 Procedure FilterPosition( position: positiontype;
62   Var filtposition: positiontype);
63 {*****}
64 {Perform a filtering action on the positions calculated, to make sure the
65 position output will be smooth.
66   Input :position
67   Output:filtered position}
68 {*****}
69
70
71 Procedure PredictPosition( position: positiontype;
72   Var predposition: positiontype);
73 {*****}
74 {Make a prediction of the position, to counteract the calculation and
75 measurement delay.
76   Input :position
77   Output:predicted position}
78 {*****}
79
80 Procedure SendPosition( position: positiontype);
81 {*****}
82 {Send the position to another device from storage or displaying, or display
83 the position on the screen.
84   Input :position
85   Output:position to the screen or a device}
86 {*****}
87
88 Function Stopcommand( command: commandtype): Boolean;
89 {*****}
90 {When a stop command is sent, this function will turn TRUE.
91   Input : commandstring
92   Output: Boolean, stop or not}
93 {*****}
94
95
96 Procedure CloseDown( alldata: alldatatype; position: positiontype);
97 {*****}
98 {Make sure the computer is back to 'normal'. Restore interruptvectors etc.
99   Input :-
100   Output:-}
101 {*****}
102
103
104 Procedure SetTimerToGPSIfNotSet;
105 {*****}
106 { Use the GPS time to set the internal timer. If the time was already set,
107 do nothing.
108   Input :-
109   Output:-}
110 {*****}

```

```

111
112
113 Implementation
114
115 Uses GPS, DGPS, MLS, Att, HDG, PosCalc, User, Miscell, crt, dos;
116
117 Const
118   Valid_tMLS      :      timetype      =      (year :0;
119   month:0;
120   day :0;
121   hour :0;
122   minute:0;
123   sec :2;
124   sec100:0);
125
126 Var
127   stopkeypressed: Boolean;
128   DGPSflag      :      flagtype;      ( Indicates DGPS mode or not)
129   DGPSmode      :      Byte;
130   allowed_error :      Double;
131   tMLS          :      timetype;
132   Old_MLS       :      MLSdatatype;
133   timeset       :      Boolean;
134   MIASlogname   :      String;
135
136
137 Procedure Init( Var alldata: alldatatype; Var position: positiontype);
138
139 Var
140   setupfile      :      Text;
141   error          :      Boolean;
142   title,
143   varname,
144   line           :      String;
145   value          :      String;
146   code           :      Integer;
147
148 Begin
149   MIASlogname:= 'c:';
150
151   DGPSflag:= True;
152   DGPSmode:= 1;
153
154   ErrorTime( tMLS);
155   With Old_MLS Do
156   Begin
157     flag:= True;
158   End;
159
160   TimeSet:= False;
161
162   With position Do
163   Begin
164     WGS84lat:= 0;
165     WGS84lon:= 0;

```

```

166     WGS84alt:= 0;
167     h:=0;
168     x:=0;
169     y:=0;
170     z:=0;
171     flag:= True;
172     integrity.flag:= True;
173   End;
174
175   alldata.mls.MLStHresPos:= position;
176   MLSantposition:= position;
177
178   If Not FileExist( MIAScfname)
179   Then Begin
180     SendUserMessage('Configfile "MIAS.CFG" not present');
181     Halt( 1);      { terminate the program }
182   End;
183
184   OpenConfigRead( setupfile, MIAScfname);
185   Repeat
186     { find MIAS part of
187     config file}
188     Readln( setupfile, title);
189   Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'MIAS'));
190
191   If Not Eof( setupfile)      { if there is more in file}
192   Then Repeat
193     Readln( setupfile, line);      { get a line}
194     Convert( line, varname, value); { extract the variable name
195                                     and value}
196     { repeat until end of file}
197     { initialise comports for
198     communication with Engine}
199
200     If ( varname = 'ALLOWED_ERROR')
201     Then Val( value, allowed_error, code);
202
203     If ( varname = 'POSITION.WGS84LAT')
204     Then Val( value, position.wgs84lat, code);
205
206     If ( varname = 'POSITION.WGS84LON')
207     Then Val( value, position.wgs84lon, code);
208
209     If ( varname = 'POSITION.WGS84ALT')
210     Then Val( value, position.wgs84alt, code);
211
212     If ( varname = 'DGPSMODE')
213     Then Val( value, dgpsmode, code);
214
215     If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84LAT')
216     Then Val( value, alldata.mls.mlsthrespos.wgs84lat, code);
217
218     If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84LON')
219     Then Val( value, alldata.mls.mlsthrespos.wgs84lon, code);
220
221     If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84ALT')

```

```

221      Then Val( value, alldata.mls.mlsthrespos.wgs84alt, code);
222
223      If ( varname = 'ALLDATA.POS_ZEROVECTOR.X')
224      Then Val( value, alldata.pos_zerovector.x, code);
225
226      If ( varname = 'ALLDATA.POS_ZEROVECTOR.Y')
227      Then Val( value, alldata.pos_zerovector.y, code);
228
229      If ( varname = 'ALLDATA.POS_ZEROVECTOR.Z')
230      Then Val( value, alldata.pos_zerovector.z, code);
231
232      If ( varname = 'ALLDATA.ANT_ZEROVECTOR.X')
233      Then Val( value, alldata.ant_zerovector.x, code);
234
235      If ( varname = 'ALLDATA.ANT_ZEROVECTOR.Y')
236      Then Val( value, alldata.ant_zerovector.y, code);
237
238      If ( varname = 'ALLDATA.ANT_ZEROVECTOR.Z')
239      Then Val( value, alldata.ant_zerovector.z, code);
240
241      If ( varname = 'ALLDATA.GPS.PRESENT')
242      Then Val( value, alldata.gps.present, code);
243
244      If ( varname = 'ALLDATA.DGPS.PRESENT')
245      Then Val( value, alldata.dgps.present, code);
246
247      If ( varname = 'ALLDATA.MLS.PRESENT')
248      Then Val( value, alldata.mls.present, code);
249
250      If ( varname = 'ALLDATA.ATT.PRESENT')
251      Then Val( value, alldata.att.present, code);
252
253      If ( varname = 'ALLDATA.HDG.PRESENT')
254      Then Val( value, alldata.hdg.present, code);
255
256      If ( varname = 'MIASLOGNAME')
257      Then MIASlogname:= value;
258      Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
259      CloseConfig( setupfile);
260
261      Convert_Pos_to_Ecef( position);
262      Convert_Pos_to_Ecef( alldata.mls.mlsthrespos);
263
264      MLSantposition:= position;
265
266      InitUser( MIASlogname);
267
268      With alldata Do
269      Begin
270          InitGPS( gps);
271          InitDGPS( dgps);
272          InitMLS( mls);
273          InitAtt( att);
274          InitHDG( hdg);
275      End;

```

```

276
277      DispFlags( alldata, position);
278 End;
279
280
281 Procedure DispFlags( alldata: alldatatype; position: positiontype);
282
283 Var
284     line          :      Commandtype;
285
286 Begin
287     line:= '';
288     With alldata Do
289     Begin
290         If Not GPS.flag Then line:= line + 'GPS oke '
291         Else line:= line + 'GPS err ' ;
292
293         If Not DGPS.flag Then line:= line + 'DGPS oke '
294         Else line:= line + 'DGPS err ' ;
295
296         If Not MLS.flag Then line:= line + 'MLS oke '
297         Else line:= line + 'MLS err ' ;
298
299         If Not Att.flag Then line:= line + 'Att oke '
300         Else line:= line + 'Att err ' ;
301
302         If Not HDG.flag Then line:= line + 'HDG oke '
303         Else line:= line + 'HDG err ' ;
304
305         If Not position.flag Then line:= line + 'Pos oke '
306         Else line:= line + 'Pos err ' ;
307
308         If ( position.integrity.flag) Then line:= line + 'int err '
309         Else line:= line + 'int oke ' ;
310         SendUserFlags( line);
311     End;
312 End;
313
314
315 Procedure GetUserCommands( Var command: commandtype);
316
317 Begin
318     GetUserMessage( command);
319 End;
320
321
322 Procedure ExecCommands( command: commandtype; alldata: alldatatype);
323
324 Var
325     substr          :      String;
326     varname,
327     value           :      String;
328     code            :      Integer;
329     tempdgpsmode    :      Byte;
330

```

```

331 Begin
332   substr:= Copy( command, 1, 4);
333
334   If ( substr = 'GPS:') And ( alldata.gps.present = 1)
335   Then ExecGPScommand( command);
336
337   If ( substr = 'MLS:') And ( alldata.mls.present = 1)
338   Then ExecMLScommand( command);
339
340   If ( substr = 'ATT:') And ( alldata.att.present = 1)
341   Then ExecAttcommand( command);
342
343   If ( substr = 'HDG:') And ( alldata.hdg.present = 1)
344   Then ExecHDGcommand( command);
345
346   substr:= Copy( command, 1, 5);
347
348   If ( substr = 'DGPS:') And ( alldata.dgps.present = 1)
349   Then ExecDGPScommand( command);
350
351   If substr = 'MIAS:'
352   Then Begin
353     substr:= Copy( command, 6, Length( command) - 5);
354     Convert( ' '+substr, varname, value);{ add space for}
355                                           { procedure convert}
356     If (varname = 'DGPSMODE')
357     Then Begin
358       Val( value, tempdgpsmode, code);
359       If code <> 0
360       Then SendUserMessage( 'Error')
361       Else dgpsmode:= tempdgpsmode;
362     End;
363   End;
364 End;
365
366 Procedure GetData( Var alldata: alldatatype);
367
368 Begin
369   With alldata Do
370   Begin
371     If ( gps.present = 1)
372     Then GetGPSdata( GPS, DGPSflag);
373
374     If ( mls.present = 1) Or           { call getmlsdata also}
375     ( dgps.present = 1)             { for dgps data}
376     Then GetMLSdata( MLS);
377
378     If ( dgps.present = 1)
379     Then GetDGPSdata( MLS, DGPS);
380
381     If Not gps.flag
382     Then Begin
383       If ( att.present = 1)
384       Then GetAttdata( Att);

```

```

386
387       If ( hdg.present = 1 )
388       Then GetHDGdata( HDG);
389     End;
390   End;
391 End;
392
393 Procedure CalcPos( Var alldata: alldatatype; Var position: positiontype);
394
395 Const
396   MaxNumOfIt =      1;                { 1 is for no iterations}
397 Var
398   old_deltaT      :      Double;
399   iter            :      Integer;
400   t,
401   result          :      timetype;
402
403 Begin
404   Case DGPSmode Of
405     1: DGPSflag:= False;              { no Diff GPS}
406     2: ;                             { only DGPS in prop cov}
407     3: DGPSflag:= True;               { always DGPS if available}
408   End;
409
410   If ( alldata.mls.present = 1)
411   Then Begin
412     CalcMLS( alldata.MLS);
413
414     If Not alldata.mls.flag           { if valid info, then adjust}
415     Then Date_and_time( tMLS);       { receive time}
416
417     Date_and_time( t);                { get current system time}
418     AddTime( tMLS, Valid_tMLS, result);
419
420     If alldata.MLS.flag And           { if no valid info and}
421     Not Later( t, result)             { no time out}
422     Then alldata.MLS:= Old_MLS        { use old info}
423     Else Old_MLS:= alldata.MLS;      { store valid info for later use}
424   End;
425
426   If ( alldata.gps.present = 1)
427   Then CalcGPS( alldata.GPS, position, DGPSflag);
428
429   If ( alldata.dgps.present = 1)
430   Then CalcDGPS( alldata);
431
432   CalcHybridPos( alldata, allowed_error, position);
433 End;
434
435 Procedure FilterPosition( position: positiontype);

```

```

441          Var filtposition: positiontype);
442
443 Begin
444 End;
445
446
447 Procedure PredictPosition( position: positiontype;
448          Var predposition: positiontype);
449 Begin
450 End;
451
452
453 Procedure SendPosition( position: positiontype);
454
455 Var
456   line      :      String;
457   dum       :      String;
458
459 Begin
460   If position.flag
461   Then Exit;
462
463   line:= '';
464   With position DO
465   Begin
466     If EcefTrueLocalFalse
467     Then Begin
468       Str( wgs84lat * 180 / pi:23, dum);
469       line:= line + 'lat = ' + dum + ' ';
470       Str( wgs84lon * 180 / pi:23, dum);
471       line:= line + 'lon = ' + dum + ' ';
472       Str( wgs84alt:23, dum);
473       line:= line + 'alt = ' + dum + ' ';
474       SendUserMessage( line);
475
476       Str( x:23, dum);
477       line:= 'x = ' + dum + ' ';
478       Str( y:23, dum);
479       line:= line + 'y = ' + dum + ' ';
480       Str( z:23, dum);
481       line:= line + 'z = ' + dum + ' ';
482       SendUserMessage( line);
483     End
484   Else Begin
485     Str( x:23, dum);
486     line:= line + 'a = ' + dum + ' ';
487     Str( y:23, dum);
488     line:= line + 'y = ' + dum + ' ';
489     Str( z:23, dum);
490     line:= line + 'z = ' + dum + ' ';
491     SendUserMessage( line);
492   End;
493 End;
494 End;
495

```

```

496
497 Function Stopcommand( command: commandtype): Boolean;
498
499 Begin
500   If Copy( command, 1, 4) = 'STOP'
501   Then stopcommand:= True
502   Else stopcommand:= False;
503 End;
504
505
506 Procedure CloseDown( alldata: alldatatype; position: positiontype);
507
508 Var
509   setupfile      :      Text;
510   Value          :      String;
511
512 Begin
513   OpenConfigWriteFirst( setupfile, MIAScfnname);
514   Writeln( setupfile, 'MIAS');
515
516   Str( allowed_error, value);
517   Writeln( setupfile, #9'allowed_error = ', value, ';');
518
519   With position Do
520   Begin
521     Str( WGS84lat, value);
522     Writeln( setupfile, #9'position.wgs84lat = ', value, ';');
523
524     Str( WGS84lon, value);
525     Writeln( setupfile, #9'position.wgs84lon = ', value, ';');
526
527     Str( WGS84alt, value);
528     Writeln( setupfile, #9'position.wgs84alt = ', value, ';');
529   End;
530   Str( dgpsmode, value);
531   Writeln( setupfile, #9'dgpsmode = ', value, ';');
532
533   With alldata.mls.MLsthrrespos Do
534   Begin
535     Str( wgs84lat, value);
536     Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84lat = ',
537             value, ';');
538
539     Str( wgs84lon, value);
540     Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84lon = ',
541             value, ';');
542
543     Str( wgs84alt, value);
544     Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84alt = ',
545             value, ';');
546   End;
547
548   With alldata.Pos_zerovector Do
549   Begin
550     Str( x, value);

```

```

551      Writeln( setupfile, #9'alldata.pos_zerovector.x = ', value, ';' );
552
553      Str( y, value );
554      Writeln( setupfile, #9'alldata.pos_zerovector.y = ', value, ';' );
555
556      Str( z, value );
557      Writeln( setupfile, #9'alldata.pos_zerovector.z = ', value, ';' );
558 End;
559
560 With alldata.Ant_zerovector Do
561 Begin
562     Str( x, value );
563     Writeln( setupfile, #9'alldata.ant_zerovector.x = ', value, ';' );
564
565     Str( y, value );
566     Writeln( setupfile, #9'alldata.ant_zerovector.y = ', value, ';' );
567
568     Str( z, value );
569     Writeln( setupfile, #9'alldata.ant_zerovector.z = ', value, ';' );
570 End;
571
572 With alldata Do
573 Begin
574     Str( gps.present, value );
575     Writeln( setupfile, #9'alldata.gps.present = ', value, ';' );
576
577     Str( dgps.present, value );
578     Writeln( setupfile, #9'alldata.dgps.present = ', value, ';' );
579
580     Str( mls.present, value );
581     Writeln( setupfile, #9'alldata.mls.present = ', value, ';' );
582
583     Str( att.present, value );
584     Writeln( setupfile, #9'alldata.att.present = ', value, ';' );
585
586     Str( hdg.present, value );
587     Writeln( setupfile, #9'alldata.hdg.present = ', value, ';' );
588 End;
589
590 Writeln( setupfile, #9'miaslogname = ', miaslogname, ';' );
591
592 CloseConfig( setupfile );
593
594 With alldata Do
595 Begin
596     If ( gps.present = 1 )
597     Then CloseGPS( gps, position );
598
599     If ( dgps.present = 1 )
600     Then CloseDGPS( dgps );
601
602     If ( mls.present = 1 )
603     Then CloseMLS( mls );
604
605     If ( att.present = 1 )

```

```

606     Then CloseAtt( att );
607
608     If ( hdg.present = 1 )
609     Then CloseHDG( hdg );
610 End;
611
612 CloseUser;
613 End;
614
615
616
617 Procedure SetTimerToGPSIfNotSet;
618
619 Var
620     gpstime      :      Longint;
621     valid        :      boolean;
622     hour,
623     minute,
624     sec,
625     sec100,
626     year,
627     month,
628     day,
629     dayofweek    :      Word;
630
631
632 Begin
633     If Not TimeSet
634     Then Begin
635         GetGPSTime( gpstime, valid );
636         If valid
637         Then Begin
638             day:= gpstime Div 86400;
639             gpstime:= gpstime Mod 86400;
640
641             hour:= gpstime div 3600;
642             gpstime:= gpstime Mod 3600;
643
644             minute:= gpstime div 60;
645             gpstime:= gpstime Mod 60;
646
647             sec:= gpstime;
648             sec100:= 0;
649
650             SetTime( hour, minute, sec, sec100 );
651             TimeSet:= True;
652
653             SendUserMessage(
654                 'Time set to User time of GPS receiver' );
655         end;
656     End;
657 End;
658
659
660 Begin

```

Page 7, listing of MIAS.PAS, date is 18-02-93, file date is 29-01-93, size is 21553 bytes.

661 End.

```

1 Unit GPS;
2
3 Interface
4
5 {$N+,E+}
6
7 Uses MIASglob, crt;
8
9 Procedure InitGps( Var GPSdata: gpsdatatype);
10 {*****}
11 {Initialise the GPS part of the system.
12   Input :-
13   Output:error; If something went wrong, error is
14   set to True}
15 {*****}
16
17
18 Procedure GetGPSdata( Var GPSdata: GPSdatatype; DGPS: Boolean);
19 {*****}
20 {Retrieve data from the GPS receiver connected to the system. Check the col-
21 lected data for age. When True, the DGPS flag indicates that DGPS mode is
22 active, and only ephemeris and pseudoranges should be valid.
23   Input :DGPS flag
24   Output:GPSdata; relevant data from the GPS
25   receiver}
26 {*****}
27
28
29 Procedure CalcGPS( Var GPSdata: GPSdatatype; position: positiontype;
30   DGPS: Boolean);
31 {*****}
32 { This procedure executes the necessary calculations for GPS. If the DGPS
33 flag is true, only the SV position and the Elevation and Azimuth to the SV
34 are calculated (For DGPS no corrections for clock, ionosphere, troposphere
35 etc are needed). If the DGPS flag is false, then all these corrections are
36 needed.
37   Input : GPSdata, ephemeris etc
38   DGPS flag
39   Output: GPSdata, SV positions}
40 {*****}
41
42
43 Procedure ExecGPScommand( command: commandtype);
44 {*****}
45 { This procedure receives a command destined for the GPS part of the MIAS
46 system. It passes the command on to the GPSreceiver-part of the system.
47   Input : command
48   Output:-}
49 {*****}
50
51
52 Procedure CloseGps( GPSdata: GPSdatatype; position: positiontype);
53 {*****}
54 {Make sure the GPS part is back to 'normal'.
55   Input :-
56
57   Output:-}
58 {*****}
59
60 Procedure GetGPStime( Var gpstime: Longint; Var valid: boolean);
61 {*****}
62 { Get the GPStime from the 'gpsint' variable. The resolution is in seconds.
63   Input :-
64   Output:GPStime and a valid boolean}
65 {*****}
66
67
68 Implementation
69
70 Uses GPSEngine, GPSscal, GPSglob, Miscell, user;
71
72 Var
73   GPSint      :      GPSinttype;
74   El_limit    :      Byte;           { satellite elevation limit}
75   HorAccFac   :      Real;          { horizontal acceleration
76                                       factor}
77   x           :      Integer;       {counter for initialising}
78
79
80 Procedure InitGps( Var GPSdata: gpsdatatype);
81
82 Var
83   setupfile   :      Text;
84   title       :      String;
85   line        :      String;
86   varname     :      String;
87   value       :      String;
88   code        :      Integer;
89   deg,
90   min         :      Double;
91   time        :      timetype;
92   position    :      positiontype;
93   x           :      Integer;       {counter for initialising}
94   error       :      Boolean;
95
96 Begin
97   error:= True;
98   GPSdata.flag:= error;
99   GPSdata.deltaT:= 0;
100  For x:= 1 To 32 Do
101    With GPSdata. prn[ x] Do
102      Begin
103        flag:= True;
104        pr := 0;
105      End;
106  If ( gpsdata.present = 0)
107  Then Exit;
108
109  OpenConfigRead( setupfile, MIAScfname);
110  Repeat

```

{ find GPS part of



```

111          config file)
112      Readln( setupfile, title);
113      Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'GPS'));
114
115      If Not Eof( setupfile)          { if there is more in file}
116      Then Repeat
117          Readln( setupfile, line);    { get a line}
118          Convert( line, varname, value); { extract the variable name
119                                          and value}
120                                          { repeat until end of file}
121          If ( varname = 'POSITION.WGS84LAT')
122          Then Val( value, position.wgs84lat, code);
123
124          If ( varname = 'POSITION.WGS84LON')
125          Then Val( value, position.wgs84lon, code);
126
127          If ( varname = 'POSITION.WGS84ALT')
128          Then Val( value, position.wgs84alt, code);
129
130          If ( varname = 'EL_LIMIT')
131          Then Val( value, el_limit, code);
132
133          If ( varname = 'HORACCFAC')
134          Then Val( value, horaccfac, code);
135      Until ( Eof( setupfile) Or ( Pos( ' ', line) = 0));
136      CloseConfig( setupfile);
137
138      InitGPSrec( error);
139
140      ExecGPSrecommand( 'GPS:RESET');
141
142      Date_and_time( time);
143      With time Do
144      Begin
145          Str( day, value);
146          value:= Copy( zeros, 1, 2 - Length( value)) + value;
147          line:= value + ' ';
148          { It contains the date and
149          { time, the latitude, longi-}
149          Str( month, value);
150          value:= Copy( zeros, 1, 2 - Length( value)) + value;
151          line:= line + value + ' ';
152          { tude and altitude of the}
153          { last position}
153          Str( year, value);
154          { don't take the 19 from }
154          value:= Copy( value, 3, 2);
155          { 19xx, only take xx}
155          value:= Copy( zeros, 1, 2 - Length( value)) + value;
156          line:= line + value + ' ';
157          { add zeros}
158          Str( hour, value);
159          value:= Copy( zeros, 1, 2 - Length( value)) + value;
160          line:= line + value;
161          { add zeros}
162          Str( minute, value);
163          value:= Copy( zeros, 1, 2 - Length( value)) + value;
164          line:= line + value + '00 ';
165          { add zeros}
166      End;

```

```

166      With position Do
167      Begin
168          deg:= Abs( Trunc( wgs84lat * 180 / pi));
169          Str( deg :2 : 0, value);
170          While value[ 1] = ' ' Do          { delete leading spaces}
171              value:= Copy( value, 2, Length( value) -1);
172          value:= Copy( zeros, 1, 2 - Length( value)) + value;
173          line:= line + value;
174          { add zeros}
175          min:= 60 * ( Abs( wgs84lat * 180 / pi) - deg);
176          Str( min :7 : 4, value);
177          While value[ 1] = ' ' Do          { delete leading spaces}
178              value:= Copy( value, 2, Length( value) -1);
179          value:= Copy( zeros, 1, 7 - Length( value)) + value;
180          line:= line + value + ' ';
181          { add zeros}
182          If deg < 0
183          Then line:= line + 'S '
184          Else line:= line + 'N ';
185
186          deg:= Abs( Trunc( wgs84lon * 180 / pi));
187          Str( deg :2 : 0, value);
188          While value[ 1] = ' ' Do          { delete leading spaces}
189              value:= Copy( value, 2, Length( value) -1);
190          value:= Copy( zeros, 1, 3 - Length( value)) + value;
191          line:= line + value;
192          { add zeros}
193          min:= 60 * ( Abs( wgs84lon * 180 / pi) - deg);
194          Str( min :7 : 4, value);
195          While value[ 1] = ' ' Do          { delete leading spaces}
196              value:= Copy( value, 2, Length( value) -1);
197          value:= Copy( zeros, 1, 7 - Length( value)) + value;
198          line:= line + value + ' ';
199          { add zeros}
200          If deg > 0
201          Then line:= line + 'E '
202          Else line:= line + 'W ';
203
204          If Abs( wgs84alt) > 99999.9          { if overflow}
205          Then value:= '99999.9'          { then take maximum value}
206          Else Str( wgs84alt :7 :1, value);
207          While value[ 1] = ' ' Do          { delete leading spaces}
208              value:= Copy( value, 2, Length( value) -1);
209          If wgs84alt < 0
210          Then value:= '-' + Copy( zeros, 1, 7 - Length( value)) +
211                      Copy( value, 2, Length( value))
212          Else value:= Copy( zeros, 1, 7 - Length( value)) + value;
213          line:= line + value;
214          { add zeros}
215      End;
216      Str( HorAccFac :4 :1, value);
217      While value[1] = ' ' Do
218          value:= Copy( value, 2, Length( value) -1);
219      value:= Copy( zeros, 1, 4 - Length( value)) + value;
220      line:= line + ' ' + value;

```

```

221 Str( el_limit: 2, value);
222 While value[1] = ' ' Do
223     value:= Copy( value, 2, Length( value) -1);
224     value:= Copy( zeros, 1, 2 - Length( value)) + value;
225     line:= line + ' ' + value;
226
227 ExecGPSreccommand( 'GPS:INIT '+ line);
228 ExecGPSreccommand( 'GPS:SEND EPHEMERIS ETC');
229
230 GPSdata.flag:= error;
231 End;
232
233
234 Procedure GetGPSdata( Var GPSdata: GPSdatatype; DGPS: Boolean);
235
236
237 Var
238     currenttime,
239     result      :      timetype;
240     x            :      Byte;
241
242 Begin
243     CollectGPSrec( GPSint);
244
245     Date_and_Time( currenttime);           { check if data valid}
246     For x:= 1 To 32 Do                     { check for every sv}
247     Begin
248         With GPSint.prn[x] Do
249         Begin
250             If Not flag                     { The flag is True if no}
251             Then Begin                     { PR was received. if no}
252                                     { PR is received, exit}
253                 AddTime( Tck, Valid_Tck, result); { also for DGPS mode}
254                 flag:= Later( currenttime, result);
255                                     { because health is trans-}
256                                     { mitted in same subframe}
257
258             If Not flag
259             Then Begin
260                 AddTime( Tephem, Valid_Tephem, result);
261                 flag:= Later( currenttime, result);
262                 If Not flag
263                 Then flag:= ( (health And $80) <> 0);
264             End;                         { check health here also}
265         End;
266     End; { end of with}
267 End; { End of for}
268
269 GPSint.flag:= True;                      { begin with assumption}
270                                           { that info is incorrect}
271 For x:= 1 To 32 Do
272     GPSint.flag:= GPSint.flag And GPSint.prn[x].flag;
273
274 If Not DGPS                             { If not dgps, these should}
275 Then With GPSint Do
276     Begin                                { be checked also}

```

```

276      Addtime( Tionos, Valid_Tionos, result);
277      flag:= flag Or Later( currenttime, result);
278      { if ionosphere is timed out}
279      { then flag should be True}
280      End;
281
282      For x:= 1 To 32 Do { update external var}
283      GPSdata.prn[x].flag:= GPSint.prn[x].flag;
284      GPSdata.flag:= GPSint.flag;
285      End;
286
287
288      Procedure CalcGPS( Var GPSdata: GPSdatatype; position: positiontype;
289      DGPS: Boolean);
290
291      Var
292      sv_id      :      Byte;
293      line,
294      dum        :      String;
295
296      Begin
297      If GPSint.flag { GPSint not valid}
298      Then Exit; { GPSint is a global
299      variable in the GPS units}
300
301      With GPSint Do
302      For sv_id:= 1 To 32 Do
303      Begin
304      If Not prn[ sv_id].flag
305      Then Begin
306      gotoxy (1,5);
307      write (prn[ sv_id].rxtime:10:2);
308      GPSdata.prn[ sv_id]. rxtime:= Round(GPSint.prn[sv_id].rxtime);
309
310      { note: in poscalc deltaT}
311      { is subtracted. Doing so}
312      { here would be obsolete}
313      Clockcorrection( sv_id, GPSint); { always correct for clock}
314      RelCorrection( sv_id, GPSint);
315      If ( Not DGPS) And ( Not position.flag)
316      Then Begin
317      L1correction( sv_id, GPSint);
318      Convert_Pos_to_WGS( position);
319      Elev_Azim( sv_id, GPSint, position);
320      Ionosphericcorrection( sv_id, position, GPSint);
321      Troposphericcorrection( sv_id, position, GPSint);}
322      End;
323
324      SVposition( sv_id, GPSint);
325      Calc_Pr( sv_id, GPSint);
326      gpsint.deltat:= gpsdata.deltat; { deltaT from position
327      calculation copied to
328      internal variable to
329      be used to correct
330      satellite position)

```

```

331         SVpos_earthadjusted( sv_id, GPSint);
332
333         GPSdata.prn[sv_id].position:= GPSint.prn[sv_id].position;
334         GPSdata.prn[sv_id].pr:= GPSint.prn[sv_id].pr;
335         GPSdata.prn[sv_id].intcarphase:=GPSint.prn[sv_id].intcarphase;
336     End;
337 End;
338 End;
339
340
341 Procedure ExecGPScommand( command: commandtype);
342
343 Begin
344     ExecGPSrecommand( command);
345 End;
346
347
348 Procedure CloseGps( GPSdata: GPSdatatype; position: positiontype);
349
350 Var
351     x      :      Byte;
352     setupfile  :      Text;
353     value      :      String;
354
355
356 Begin
357     OpenConfigWrite( setupfile, MIAScfgname);
358     Writeln( setupfile, 'GPS');
359
360     With position Do
361     Begin
362         Str( WGS84lat, value);
363         Writeln( setupfile, #9'position.wgs84lat = ', value, ');');
364
365         Str( WGS84lon, value);
366         Writeln( setupfile, #9'position.wgs84lon = ', value, ');');
367
368         Str( WGS84alt, value);
369         Writeln( setupfile, #9'position.wgs84alt = ', value, ');');
370     End;
371
372     Str( el_limit, value);
373     Writeln( setupfile, #9'el_limit', ' = ', el_limit, ');');
374
375     Str( horaccfac, value);
376     Writeln( setupfile, #9'horaccfac', ' = ', horaccfac, ');');
377
378     CloseConfig( setupfile);
379
380     CloseGPSrec;
381 End;
382
383
384 Procedure GetGPStime( Var gpstime: Longint; Var valid: boolean);
385

```

```

386 Var
387     x      :      Byte;
388
389 Begin
390     If gpsint.flag = true
391     Then Begin
392         valid:= False;
393         Exit;
394     End
395     Else Begin
396         x:=1;
397         While ( gpsint.prn[x].flag = true) And
398             ( x < 32) Do
399             Inc( x);
400         If x <= 32
401         Then Begin
402             gpstime:= Round( gpsint.prn[x].rxtime);
403             valid:= True;
404         End
405         Else valid:= False;
406     End;
407 End;
408
409
410 Begin                                     ( initialising part)
411     GPSint.flag:= True;
412     GPSint.numofsat := 0;
413     ErrorTime( GPSint.Tionos);
414     For x:= 1 To 32 Do
415     With GPSint.prn[x] Do
416     Begin
417         flag:= True;
418         Ek:=0;
419         pr:= 0;
420         ErrorTime( Tck);
421         ErrorTime( Tephem);
422     End;
423
424     EL_limit:= 0;
425     HorAccFac:= 0;
426 End.

```

```

1 Unit MLS;
2
3 {$N+,E+}
4
5 Interface
6
7 Uses MIASglob;
8
9 Procedure InitMLS( Var MLSdata: MLSdatatype);
10 {*****}
11 {Initialise the MLS sensor. If something went wrong, the error flag is set
12  to the value True.      Input :-
13                          Output:error}
14 {*****}
15
16
17 Procedure GetMLSdata( Var MLSdata: MLSdatatype);
18 {*****}
19 {Retrieve MLS data from the MLS sensor.
20      Input :-
21      Output:MLSdata}
22 {*****}
23
24
25
26 Procedure CalcMLS( Var MLSdata: MLSdatatype);
27 {*****}
28 {This procedure performs the necessary MLS calculations.
29      Input : MLSdata
30      Output: MLSdata.}
31 {*****}
32
33
34 Procedure ExecMLScommand( command: commandtype);
35 {*****}
36 {This procedure sends commands to the MLS receiver.
37      Input : command
38      Output: -}
39 {*****}
40
41
42 Procedure CloseMLS( MLSdata: MLSdatatype);
43
44 {*****}
45 {Closedown the MLS sensor. Input :-
46      Output:-}
47 {*****}
48
49
50 Implementation
51
52
53 Uses MLSbendix, MLScglob, Miscell, crt, user;
54
55 Var

```

```

56 MLSint      :      MLSinttype;
57
58
59 Procedure InitMLS( Var MLSdata: MLSdatatype);
60
61 Var
62     setupfile  :      Text;
63     title      :      String;
64     line        :      String;
65     varname     :      String;
66     value       :      String;
67     code        :      Integer;
68     error       :      Boolean;
69
70 Begin
71     error:= True;
72     MLSdata.flag:= error;
73
74     If ( mlsdata.present = 0)
75     Then Exit;
76
77     OpenConfigRead( setupfile, MIAScfname);
78     Repeat                                     { find MLS part of
79                                                config file}
80         Readln( setupfile, title);
81     Until (EOF( setupfile) OR ( Copy( title, 1, 3) = 'MLS'));
82
83     If Not Eof( setupfile)                    { if there is more in file}
84     Then Repeat
85         Readln( setupfile, line);             { get a line}
86         Convert( line, varname, value);       { extract the variable name
87                                                and value}
88         { repeat until end of file}
89         { initialise comports for
90           communication with Engine}
91     Until ( Eof( setupfile) Or ( Pos( ' ', line) = 0));
92     CloseConfig( setupfile);
93
94     InitMLSrec( error);
95     MLSdata.flag:= error;
96 End;
97
98
99 Procedure GetMLSdata( Var MLSdata: MLSdatatype);
100
101 Var
102     result,
103     currenttime :      Timetype;
104
105 Begin
106     CollectMLSrec( MLSint);
107
108     Date_and_Time( currenttime);              { check if data valid}
109
110     With MLSint Do

```

```

111 Begin
112     AddTime( Bas1_time, Valid_Bas1, result);
113     Bas1_flag:= Later( currenttime, result);
114
115     AddTime( Bas2_time, Valid_Bas2, result);
116     Bas2_flag:= Later( currenttime, result);
117
118     AddTime( Bas3_time, Valid_Bas3, result);
119     Bas3_flag:= Later( currenttime, result);
120
121     AddTime( Bas4_time, Valid_Bas4, result);
122     Bas4_flag:= Later( currenttime, result);
123
124     AddTime( Bas5_time, Valid_Bas5, result);
125     Bas5_flag:= Later( currenttime, result);
126
127     AddTime( Bas6_time, Valid_Bas6, result);
128     Bas6_flag:= Later( currenttime, result);
129
130     AddTime( AuxA1_time, Valid_AuxA1, result);
131     AuxA1_flag:= Later( currenttime, result);
132
133     AddTime( AuxA2_time, Valid_AuxA2, result);
134     AuxA2_flag:= Later( currenttime, result);
135
136     AddTime( AuxA3_time, Valid_AuxA3, result);
137     AuxA3_flag:= Later( currenttime, result);
138
139     AddTime( AuxA4_time, Valid_AuxA4, result);
140     AuxA4_flag:= Later( currenttime, result);
141
142     AddTime( AuxB_time, Valid_AuxB, result);
143     AuxB_flag:= Later( currenttime, result);
144
145     AddTime( AuxC_time, Valid_AuxC, result);
146     AuxC_flag:= Later( currenttime, result);
147
148     flag:= (Elangle_flag And Azangle_flag) Or
149            Bas1_flag Or Bas2_flag Or Bas3_flag Or Bas4_flag Or
150            Bas6_flag Or AuxA1_flag Or AuxA2_flag Or AuxA3_flag;
151     flag:= flag And ( BazAngle_flag Or Bas4_flag Or Bas5_flag Or
152                    Bas6_flag Or AuxA3_flag Or AuxA4_flag);
153     { According to Annex 10
154       p 61, §3.11.5.4, When
155       BAZ is provided, AuxA4
156       And Bas5 should be trans-
157       mitted in the Approach
158       region also. This is not
159       implemented here}
160
161     MLSdata.Elangle:= Elangle;
162     MLSdata.Azangle:= Azangle;
163     MLSdata.BAZangle:= Bazangle;
164     MLSdata.DMErange:= DMErange;
165
166     MLSdata.leftclr:= leftclr;
167     MLSdata.rightclr:= rightclr;
168     MLSdata.ElantInUse:= ElantInUse;
169     MLSdata.AzantInUse:= AzantInUse;
170     MLSdata.AuxB:= AuxB;
171     MLSdata.AuxC:= AuxC;
172     MLSdata.Elangle_flag:= Elangle_flag;
173     MLSdata.AZangle_flag:= AZangle_flag;
174     MLSdata.BAZangle_flag:= BAZangle_flag;
175     MLSdata.DME_flag:= DME_flag;
176     MLSdata.AuxB_flag:= AuxB_flag;
177     MLSdata.AuxC_flag:= AuxC_flag;
178
179
180     MLSdata.flag:= flag;
181 End;
182 End;
183
184
185 Procedure CalcMLS( Var MLSdata: MLSdatatype);
186
187 Begin
188     With MLSdata Do
189     Begin
190         { calculate the el, az and}
191         { baz position referenced}
192         { to the MLS datum point}
193         { MLSint is a global var-}
194         { ible in the MLS units}
195
196         With Azpos Do
197         If ( Not MLSint.AuxA1_flag)
198         Then Begin
199             x:= -1 * MLSint.AuxA1.Az2MLSdatdist;
200             y:= MLSint.AuxA1.Azoff;
201             z:= 0;
202             Azpos_flag:= False;
203         End;
204
205         With Elpos Do
206         If Not MLSint.AuxA2_flag
207         Then Begin
208             x:= 0;
209             y:= MLSint.AuxA2.Elloff;
210             z:= MLSint.AuxA2.ElHeight;
211             Elpos_flag:= False;
212         End;
213
214         With Bazpos Do
215         If ( Not MLSint.AuxA4_flag)
216         Then Begin
217             x:= MLSint.AuxA4.Baz2MLSdatdist;
218             y:= MLSint.AuxA4.Bazoff;
219             z:= 0;
220             Bazpos_flag:= False;
221
222         End;
223
224         With DMEpos Do
225         If ( Not MLSint.AuxA3_flag)
226         Then Begin
227             x:= -1 * MLSint.AuxA3.DME2MLSdatdist;
228             y:= MLSint.AuxA3.DMEoff;
229
230         End;
231     End;
232 End;

```

```

221         z:= 0;
222         DMEpos_flag:= False;
223     End;
224     If (Not MLSint.ELangle_flag) And (MLSint.bas2.elstat = 1)
225     Then Begin
226         ELangle:= MLSint.ELangle;
227         ELangle_flag:= False;
228     End
229     Else ELangle_flag:= True;
230
231     If (Not MLSint.AZangle_flag) And (MLSint.bas2.azstat = 1) And
232     (Not MLSint.AuxA1_flag)
233     Then Begin
234         Azangle:= MLSint.Azangle + MLSint.AuxA1.AzAlignRun;
235         Azangle_flag:= False;
236     End
237     Else Azangle_flag:= True;
238
239     If (Not MLSint.BAZangle_flag) And (MLSint.bas2.bazstat = 1) And
240     (MLSint.bas5.bazstat = 1) And (Not MLSint.AuxA4_flag)
241     Then Begin
242         Bazangle:= MLSint.Bazangle - MLSint.AuxA4.BazAlignRun;
243         BAZangle_flag:= False;
244     End
245     Else BAZangle_flag:= True;
246
247     If (Not MLSint.DME_flag) And (MLSint.bas2.dmestat > 0)
248     Then Begin
249         DMErange:= MLSint.DMErange;
250         DME_flag:= False;
251     End
252     Else DME_flag:= True;
253
254     flag:= ( ELangle_flag Or Elpos_flag) And
255     ( Azangle_flag Or Azpos_flag) And
256     ( Bazangle_flag Or Bazpos_flag) And
257     ( DME_flag Or DMEpos_flag);    { set the MLS flag}
258
259 { NOTE: this is magnetic heading}
260     If ( Not MLSint.Bas4_flag) And (Not MLSint.AuxA1_flag)
261     Then Begin
262         Runwayhdg:= MLSint.Bas4.AzMagOr -
263             MLSint.AuxA1.AzAlignRun + 180;
264         If Runwayhdg >= 360
265         Then Runwayhdg:= Runwayhdg - 360;
266         Runwayhdg_flag:= False;
267     End
268     Else Runwayhdg_flag:= True;
269 End;
270 End;
271
272
273 Procedure ExecMLScommand( command: commandtype);
274
275 Begin

```

```

276     ExecMLSrecommand( command);
277 End;
278
279
280 Procedure CloseMLS( MLSdata: MLSdatatype);
281
282 Begin
283     CloseMLSRec;
284 End;
285
286
287 Begin
288     With MLSint Do
289     Begin
290         Bas1_flag:= True;
291         Bas2_flag:= True;
292         Bas3_flag:= True;
293         Bas4_flag:= True;
294         Bas5_flag:= True;
295         Bas6_flag:= True;
296         AuxA1_flag:= True;
297         AuxA2_flag:= True;
298         AuxA3_flag:= True;
299         AuxA4_flag:= True;
300         AuxB_flag:= True;
301         AuxC_flag:= True;
302
303         ELangle_flag:= True;
304         AZangle_flag:= True;
305         BAZangle_flag:= True;
306         DME_flag:= True;
307         discretess_flag:= True;
308         leftclr:= True;
309         rightclr:= True;
310         flag:= True;
311
312         ErrorTime( Bas1_time);
313         ErrorTime( Bas2_time);
314         ErrorTime( Bas3_time);
315         ErrorTime( Bas4_time);
316         ErrorTime( Bas5_time);
317         ErrorTime( Bas6_time);
318         ErrorTime( AuxA1_time);
319         ErrorTime( AuxA2_time);
320         ErrorTime( AuxA3_time);
321         ErrorTime( AuxA4_time);
322         ErrorTime( AuxB_time);
323         ErrorTime( AuxC_time);
324     End;
325 End.

```

```

1 Unit GPSEngine;
2 {*****}
3 { This unit provides procedures to be used by the GPS unit. The procedures
4   are taylor made for the GPSEngine from Magnavox.}
5 {*****}
6
7
8 Interface
9
10 {$N+,E+}
11
12 Uses MIASglob, GPSglob, crt;
13
14
15 Procedure InitGPSrec( Var error: Boolean);
16 {*****}
17 {Initialise the GPS receiver. If something went wrong, error := True.
18   Input :-
19   Output:error}
20 {*****}
21
22
23 Procedure CollectGPSrec( Var GPSint: GPSinttype);
24 {*****}
25 {get characters from buffers. Synchronise with the delimiters 'LF' and '$'.
26   Check for correct header and process the information to the variable GPSint
27   Input :-
28   Output:GPSint}
29 {*****}
30
31
32 Procedure ExecGPSrecCommand( command: commandtype);
33 {*****}
34 { Convert a GPScommand, to a commandstring for the GPS Engine and send
35   the command using the IWRITECOM routine in COMM_.int. The commands
36   here start with 'GPS:'.
37   Input : command
38   Output: commandstring on comport}
39 {*****}
40
41
42 Procedure CloseGPSrec;
43 {*****}
44 {Make sure the GPS receiver is back to normal. Restore interruptvectors etc
45   Input :-
46   Output:-}
47 {*****}
48
49
50 Implementation
51
52
53 Uses User, Miscell, com_4;(comdisc)
54
55

```

```

56 Const
57     LF      =    #10;{ linefeed}
58
59 Var
60     port0,
61     port1   :      Byte;  { contains comports for port0 and port1
62                           of the GPSEngine}
63     x, y    :      Shortint; {counter}
64     two2power :      Array[ 0..55] Of Double;
65     sv_id   :      Byte;  { contains the satellite identity number
66                           for ephemeris transmissions}
67     Tsv_id,
68     Valid_Tsv_id :      timetype;{Time that sv_id should be valid}
69     Tpr,
70     Valid_Tpr  :      timetype;{ time that a set is valid, for timeout}
71     old_user_ms :      Real;  { remember old time of measurement}
72     svccount   :      Shortint;{ count the number of PR yet received}
73     tempGPSint :      GPSinttype;{ internal var containing all info received}
74     completeinfo :      Byte;  { contains info for transfer of completeinfo}
75                           { or partial info if available}
76                           { 0 for partial, 1 for complete}
77
78 {----- start procedure initgpsrec-----}
79 Procedure InitGPSrec( Var error: Boolean);
80
81 Var
82     title,
83     line      :      String;
84     code      :      Integer;
85     varname,
86     value     :      String;
87     setupfile :      Text;
88
89 Begin
90
91     { set default values}
92     port1 := 1; { Engine port 1 is com 1}
93     port0 := 2; { Engine port 0 is com 2}
94     error:= True;
95     OpenConfigRead( setupfile, MIAScfigname);
96     Repeat
97         Readln( setupfile, title);
98     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'GPSENGINE'));
99
100    If Not Eof( setupfile) { if there is more in file}
101    Then Repeat
102        Readln( setupfile, line); { get a line}
103        Convert( line, varname, value); { extract the variable name
104                                         and value}
105
106        If (varname = 'PORT0')
107        Then Val( value, port0, code);
108
109        If ( varname = 'PORT1')
110        Then Val( value, port1, code);

```

```

111      If ( varname = 'COMPLETEINFO')
112      Then Val( value, completeinfo, code);
113      Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
114      { repeat until end of file}
115      CloseConfig( setupfile);
116
117      { initialise comports for
118      communication with Engine}
119      Setupcomport( port0, Ord( B4800), 8, Ord( None), 1);
120      Setupcomport( port1, Ord( B4800), 8, Ord( None), 1);
121      { empty receive and trans-
122      mit buffers}
123      Emptybuffer( port0, True);
124      Emptybuffer( port1, True);
125      { set interrupt vectors}
126      Installint( port0); { save old interrupt}
127      Installint( port1); { vectors}
128
129      error:= False;
130 End;
131 {----- end procedure initgpsrec-----}
132
133 {----- start procedure CollectGPSrec-----}
134 Procedure CollectGPSrec( Var GPSint: GPSinttype);
135
136 {----- start included procedures CollectGPSrec-----}
137 Procedure Collectport0( Var GPSint: GPSinttype);
138 {*****}
139 { This procedure collects data from port0 of the GPS ENGINE receiver.
140      Input : data from receive buffer
141      Output: updated GPSint.}
142 {*****}
143
144 {----- start included procedures collectport0-----}
145 Procedure StatusReport( rec: String; Var GPSint: gpsinttype);
146 {*****}
147 { This procedure reads the number of satellites being tracked}
148 {      Input : string from GPSEngine
149      Output: number of satellites being tracked}
150 {*****}
151
152 Var
153     value :      String;
154     code  :      Integer;
155
156 Begin
157     value:= Copy( rec, 12, Length( rec) -11);{ select everything
158      { but the header}
159     For x:= 1 To 2 Do
160     Begin
161         While value[1] <> ',' Do
162             value:= Copy( value, 2, Length( value) -1);
163         value:= Copy( value, 2, Length( value) - 1);
164     End;
165

```

```

166     value:= Copy( value, 1, 1);
167     Val( value, GPSint.numofsat, code);
168
169     If code <> 0
170     Then GPSint.numofsat := -1;
171 End;
172 {----- end included procedures collectport0-----}
173
174 Var
175     rec,
176     header      :      String;
177     recnum,
178     code        :      Integer;
179     time        :      timetype;
180     part        :      String;
181
182 {-----Start collectport0-----}
183 Begin
184     GPSint:= tempGPSint; { begin collectport0}
185
186     If (charsinbuff( port0) > 80) { wait until string long}
187     Then Begin { enough to be valid}
188         rec:= Getcharbuff( port0); { get on char from buffer}
189         If ( rec = '$') { if begin of recordstring}
190         Then Begin
191             Repeat { read chars from buffer}
192                 rec:= rec + Getcharbuff( port0);
193             Until ( Pos( LF, rec) <> 0) Or
194                 ( charsinbuff( port0) = 0);
195                 { until a line feed found}
196                 { or no more chars in buff}
197             If ( charsinbuff( port0) = 0)
198             Then Exit; { if no lf found, exit}
199
200             { first part of string must}
201             { be $PMVSG for Engine}
202             header:= Copy( rec, 1, 7);
203             If ( header = '$PMVXG,')
204             Then Begin { next part is recordnumber}
205                 gotoxy( 1, 15);
206                 write( rec);
207                 SaveEquipmentMessage( 'RG: ' + rec);
208                 header:= Copy( rec, 8,3);
209                 { convert string to value}
210                 Val( header, recnum, code);
211                 { if no error converting}
212                 If (code = 0)
213                 Then Case recnum Of
214                     0: StatusReport( rec, GPSint);
215                 End
216                 Else; { what if error converting?}
217             End
218             Else; { what if $PMVXG not found?}
219         End
220         Else While (( rec <> LF) And

```



```

221      ( Charsinbuff( port0) > 0)) Do
222          { if not begin of record}
223          { read buffer until lf}
224          rec:= getcharbuff( port0);
225      End;
226      tempGPSint:= GPSint;
227  End;
228  {-----end collectport0-----}
229
230  Procedure Collectport1( Var GPSint: GPSinttype);
231  {*****}
232  { This procedure collects data from port1 of the GPS ENGINE receiver.
233    Input : data from receive buffer
234    Output: updated GPSint.}
235  {*****}
236
237  Type
238      temptype      =      Array[1..24] Of Byte;
239
240  {----- start included procedures collectport1-----}
241  Procedure Conv_ASCII_2_Val( rec: String; Var temp: temptype;
242    Var error: Boolean);
243  {*****}
244  { This procedure convert a string with pairs of ascii characters
245    to byte values. This string should contain 24 pairs of ascii char-
246    acters with the 'values' '0'..'9' or 'A'..'F'. The first pair should
247    begin at position 6 of the string. When the data is erroneous, then
248    the error flag is set True.
249    Input : received string
250    Output: array with 24 bytes
251    error when flag}
252  {*****}
253
254  Var
255      x, y          :      Integer;
256      value         :      Byte;
257      dum           :      Char;
258      code          :      Integer;
259
260  Begin
261      error:= False;
262          { initialise errorflag}
263
264      For x:= 0 To 23 Do
265          { count 24 sets of }
266          { characters}
267          { reset value}
268          { reset almanac}
269          { convert two characters to}
270          { value}
271          dum:= rec[6 + x * 3 + (1 - y)]; { get digit starting at }
272          { position 6}
273          If Not ( dum in ['0'..'9', 'A'..'F'])
274              { digit is hexadecimal}
275              Then Begin
276                  error:= True;
277
278          Exit;
279          { error in received record}
280      End;
281
282      Case dum Of
283          '0'..'9': Val( dum, value, code);
284          'A'..'F': value:= Ord( dum) - Ord( 'A') + 10;
285      End;
286
287      value:= Round(value * two2power[ y * 4]);{ first digit is 16's}
288          { second digit is 1's}
289      temp[ x+1] := temp[ x+1] + value;
290  End;
291
292  End;
293          { complete line converted +}
294          { stored}
295
296  Function Scale( temparray: temptype; pointer, startbit,
297    nr_of_bits: Byte): Double;
298  {*****}
299  { The function takes one or more bytes from an array called tempalmanac
300    and converts it into a value. Pointer indicates the first byte from
301    the array to be used. Startbit indicates the number of the first bit
302    to be used. The MSB has number 0, increasing to the LSB's. Nr_of_bits
303    indicates the number of bits to be used.
304    WARNING: THIS FUNCTION WILL ONLY WORK A MAXIMUM OF 32 BITS, BECAUSE
305    OF THE ROUND FUNCTION.
306    Input : array with bytes
307    pointer for first byte
308    startbitnumber
309    number of bits
310    Output: Value of the number indicated with
311    the input variables}
312  {*****}
313
314  Var
315      temp          :      Double;
316      temppointer   :      Byte;
317      x             :      Byte;
318      leftoverbits  :      Byte;
319
320  Begin
321      temppointer:= pointer; { save the pointer in the array here}
322      temp:= temparray[ temppointer];
323      Inc( temppointer); { take the pointer'th byte from the array}
324      While (startbit + nr_of_bits) - ((pointer - temppointer) * 8) > 0 Do
325          { repeat this, until enough bytes are taken}
326          Begin
327              temp:= temp * 256 + temparray[ temppointer];
328              Inc( temppointer); { give every byte its position ref weight}
329          End;
330
331          { Delete MSB's that should not be used}
332          { x is number of the bits that should not be used}
333          { subtract the values indicated by those bits}
334
335      If startbit > 0
336      Then For x:= 0 To startbit - 1 Do
337          If ( temp >= two2power[ (pointer - temppointer) * 8]

```

```

386                                     (ms))
387     chnl_ms,                          { channel time in GPS
388                                     receiver (ms))
389     phi,                             { integrated carrier phase
390                                     l1 wavelengths)
391     phi_frac,                        { idem, fractional. LSB=(l1
392                                     wavelength)/256)
393     code                             { raw code offset, l1 wave-
394                                     lengths)
395                                     :      Extended;
396     prn                              { prn-code of the satellite)
397     fault                            :      Integer;
398     subrec                           :      String;
399
400     user_time,                       { user time in GPS receiver (s))
401     channel_time                     { channel time in GPS receiver (s))
402                                     :      Extended;
403 value,
404 lineout        :      String;
405
406 Begin
407     subrec:= Copy( rec, 9, 2);       { grab satellite prn}
408     Val( subrec, prn, fault);
409     If ( fault <> 0)                  { error converting ascii}
410     Then Exit;                      { to number}
411
412     subrec:= Copy( rec, 12, 9);      { grab user_ms}
413     Val( subrec, user_ms, fault);
414     If ( fault <> 0)                  { error converting ascii}
415     Then Exit;                      { to number}
416
417     subrec:= Copy( rec, 22, 9);      { grab channel_ms}
418     Val( subrec, chnl_ms, fault);
419     If ( fault <> 0)                  { error converting ascii}
420     Then Exit;                      { to number}
421
422     subrec:= Copy( rec, 32, 10);     { grab phi}
423     Val( subrec, phi, fault);
424     If ( fault <> 0)                  { error converting ascii}
425     Then Exit;                      { to number}
426
427     subrec:= Copy( rec, 43, 6);      { grab code}
428     Val( subrec, code, fault);
429     If ( fault <> 0)                  { error converting ascii}
430     Then Exit;                      { to number}
431
432     subrec:= Copy( rec, 50, 4);      { grab phi_frac}
433     Val( subrec, phi_frac, fault);
434     If ( fault <> 0)                  { error converting ascii}
435     Then Exit;                      { to number}
436
437     If (user_ms = old_user_ms)
438     Then Begin
439         user_time:= user_ms/1000;   { calculate receive and
440                                     transmit times,
```

```

441             see MAGNAVOX guide}
442
443         With GPSint.prn[ prn] Do
444         Begin
445             intcarphase:= ( phi_frac / 256);
446             intcarphase:= intcarphase + phi;
447
448             txtime:= ( chnl_ms / 1000) +
449                 ( intcarphase / l1freq) +
450                 ( code / l1freq);
451             rxtime:= user_time;
452             flag:= False;
453
454         End;
455 { This is made a comment, to make the program work better with a disc.
456 In that case, it may assume a wrong number of satellites being tracked
457 giving erroneous results. By not incrementing 'svcount', the errors can
458 be prevented. The complete cycle of measurements is noticed as soon as
459 the next cycle begins.)
460         Inc( svcount);
461     End
462 Else Begin
463     Restore_buffer( port1, LF+rec);{ put received string back
464                                     for later use}
465     old_user_ms:= user_ms;
466     svcount:= GPSint.numofsat;
467
468     Date_and_time( Tpr);
469 End; { End of procedure rawdata}
470
471
472
473 Procedure Ionoscor( rec: String; Var GPSint: GPSinttype);
474 {*****}
475 { This procedure converts a received Engine string, containing Iono-
476 spheric information, to variables. It takes one, two or three bytes
477 from the tempionos, converts this to a value and adds an exponent.
478 The information used, can be found in Appendix 3 to Annex A to
479 STANAG 4249, 1 August 1990.
480     Input : received recordnumber
481           received string
482     Output: updated almanac in GPSint}
483 {*****}
484
485 Var
486     tempionos      :      temptype;
487     error          :      Boolean;
488
489 Begin
490     Conv_ASCII_2_val( rec, tempionos, error);
491     If error
492     Then Exit;
493     With GPSint.iono Do
494     Begin
495         alfa0:= Twoscomplement( Scale( tempionos, 2, 0, 8), 8);
496
497         alfa0:= alfa0 / two2power[ 30];
498
499         alfa1:= Twoscomplement( Scale( tempionos, 3, 0, 8), 8);
500         alfa1:= alfa1 / two2power[ 27];
501
502         alfa2:= Twoscomplement( Scale( tempionos, 4, 0, 8), 8);
503         alfa2:= alfa2 / two2power[ 24];
504
505         alfa3:= Twoscomplement( Scale( tempionos, 5, 0, 8), 8);
506         alfa3:= alfa3 / two2power[ 24];
507
508         beta0:= Twoscomplement( Scale( tempionos, 6, 0, 8), 8);
509         beta0:= beta0 * two2power[ 11];
510
511         beta1:= Twoscomplement( Scale( tempionos, 7, 0, 8), 8);
512         beta1:= beta1 * two2power[ 14];
513
514         beta2:= Twoscomplement( Scale( tempionos, 8, 0, 8), 8);
515         beta2:= beta2 * two2power[ 16];
516
517         beta3:= Twoscomplement( Scale( tempionos, 9, 0, 8), 8);
518         beta3:= beta3 * two2power[ 16];
519     End; { End of with}
520     Date_and_time( GPSint.Tionos);
521 End; { End of procedure Ionoscor}
522
523
524 Procedure Get_SV_id( rec: String; Var sv_id: Byte);
525 {*****}
526 { This procedure is needed, because the gpsengine sends the ephemeris
527 in several records. Only the first record contains the SV-id of the
528 SV from which the ephemeris originates
529     Input : received string
530     Output: satellite identity}
531 {*****}
532
533 Var
534     subrec      :      String;
535     code        :      Integer;
536
537 Begin
538     subrec:= Copy( rec, 6, 3);
539     Val( subrec, sv_id, code);
540     If ( code <> 0)
541     Then sv_id:=0;
542
543     Date_and_time( Tsv_id);
544 End;
545
546 Procedure Clockinfo( rec: String; sv_id: Byte;
547                     Var GPSint: GPSinttype);
548 {*****}
549 { This procedure converts received Engine strings, containing Clock-
550 information per satellite, to variables. It takes one, two or three

```

```

551     bytes from the tempclock, converts this to a value and adds an
552     exponent. The information used, can be found in Appendix 3 to Annex
553     A to STANAG 4249, 1 August 1990.
554         Input : received string
555                 satellite identity
556         Output: updated clock info per SV in GPSint)
557 {*****}
558 Var
559     tempclock      :      temptype;
560     error          :      Boolean;
561     sumtime,
562     time           :      timetype;
563     dum            :      Double;
564
565 Begin
566     With GPSint Do
567     Begin
568         Date_and_Time( time);          { get current system time}
569         Addtime( Tsv_id, Valid_Tsv_id, sumtime);
570         If (sv_id = 0) Or Later( time, sumtime)
571             { if Satellite identity
572              number is 0, or the infor-
573              mation is timed out, than
574              no valid information}
575         Then Begin
576             ErrorTime( Tsv_id);
577             Exit;
578         End;
579     End;
580
581     Conv_ASCII_2_val( rec, tempclock, error);
582     If error
583     Then Exit;
584
585     With GPSint.prn[ sv_id].clock Do
586     Begin
587         Tgd:= Twoscomplement( Scale( tempclock, 15, 0, 8), 8);
588         Tgd:= Tgd / two2power[ 31];
589
590         toc:= Scale( tempclock, 17, 0, 16);
591         toc:= Round( toc * two2power[ 4]);
592         If toc > 604784          { check on range}
593         Then Begin
594             ErrorTime( GPSint.prn[ sv_id].Tck);
595             Exit;
596         End;
597
598         af2:= Twoscomplement( Scale( tempclock, 19, 0, 8), 8);
599         af2:= af2 / two2power[ 55];
600
601         af1:= Twoscomplement( Scale( tempclock, 20, 0, 16), 16);
602         af1:= af1 / two2power[ 43];
603
604         af0:= Twoscomplement( Scale( tempclock, 22, 0, 22), 22);
605         af0:= af0 / two2power[ 31];

```

```

606
607     dum:= Scale( tempclock, 3, 6, 2) * 256;
608     IODC:= Integer( Round(dum));
609     dum:= Scale( tempclock, 16, 0, 8);
610     IODC:= IODC + Integer( Round(dum));
611 End;      { End with}
612 With GPSint.prn[ sv_id] Do
613 Begin
614     health:= Round( Scale( tempclock, 3, 0, 6));
615                                     { set the 6 LSB's of
616                                     health by health from
617                                     SV record}
618     If Round( Scale( tempclock, 3, 0, 1)) = 0
619     Then health:= ( health And $1F) { healthy force zero's}
620     Else health:= ( health Or $E0); { unhealthy force one's}
621                                     { use MSB from SV record
622                                     to set health to 'alldata
623                                     bad' or 'all data ok'}
624
625     Date_and_Time( Tck);
626 End;      { End with}
627 End;      { End procedure clockinfo}
628
629 Procedure Ephemeris( recnum: Integer; rec: String; sv_id: Byte;
630                     Var GPSint: GPSinttype);
631 {*****}
632 { This procedure converts received Engine strings, containing Ephem-
633   eris information, to variables. It takes one, two or three bytes
634   from the tempephem, converts this to a value and adds an exponent.
635   The information used, can be found in Appendix 3 to Annex A to
636   STANAG 4249, 1 August 1990.
637         Input : received recordnumber
638                 received string
639                 satellite identity
640         Output: updated ephemeris in GPSint)
641 {*****}
642 Var
643     tempephem      :      temptype;
644     error          :      Boolean;
645     time,
646     sumtime        :      timetype;
647
648 Begin
649     Date_and_Time( time);          { get current system time}
650     Addtime( Tsv_id, Valid_Tsv_id, sumtime);
651     If (sv_id = 0) Or Later( time, sumtime)
652         { if Satellite identity
653          number is 0, or the infor-
654          mation is timed out, than
655          no valid information}
656     Then Begin
657         ErrorTime( Tsv_id);
658         Exit;
659     End;
660

```

```

661      Conv_ASCII_2_val( rec, tempephem, error);
662      If error
663      Then Exit;
664
665      With GPSint.prn[ sv_id].ephemeris Do
666      Begin
667          Case recnum Of
668          202: Begin
669              IODE:= tempephem[ 1];
670
671              Crs:= Twoscomplement( Scale( tempephem, 2, 0, 16), 16);
672              Crs:= Crs / two2power[ 5];
673
674              deltan:= Twoscomplement( Scale( tempephem, 4, 0, 16), 16);
675              deltan:= deltan / two2power[ 43];
676
677              Mo:= Twoscomplement( Scale( tempephem, 6, 0, 32), 32);
678              Mo:= Mo / two2power[ 31];
679
680              Cuc:= Twoscomplement( Scale( tempephem, 10, 0, 16), 16);
681              Cuc:= Cuc / two2power[ 29];
682
683              e:= Scale( tempephem, 12, 0, 32);
684              e:= e / two2power[ 33];
685              If e > 0.03          { Check on range}
686              Then Begin
687                  ErrorTime( GPSint.prn[ sv_id].Tephem);
688                  Exit;
689              End;
690
691              Cus:= Twoscomplement( Scale( tempephem, 16, 0, 16), 16);
692              Cus:= Cus / two2power[ 29];
693
694              Asqrt:= Scale( tempephem, 18, 0, 32);
695              Asqrt:= Asqrt / two2power[ 19];
696
697              toe:= Scale( tempephem, 22, 0, 16);
698              toe:= toe * two2power[ 4];
699              If toe > 604784      { Check on range}
700              Then Begin
701                  ErrorTime( GPSint.prn[ sv_id].Tephem);
702                  Exit;
703              End;
704
705              Date_and_Time( GPSint.prn[ sv_id].Tephem);
706              End; { End of Case 202}
707          203: Begin
708              If (tempephem[22] <> IODE) {new ephemeris being}
709              Then Begin {uploaded. Subframe 2}
710                  ErrorTime( GPSint.prn[ sv_id].Tephem);
711                  Exit {ready, subframe 3 not}
712                  End; {don't use this sv now}
713
714              Cic:= Twoscomplement( Scale( tempephem, 1, 0, 16), 16);
715              Cic:= Cic / two2power[ 29];
716
717              omegao:= Twoscomplement( Scale( tempephem, 3, 0, 32), 32);
718              omegao:= omegao / two2power[ 31];
719
720              Cis:= Twoscomplement( Scale( tempephem, 7, 0, 16), 16);
721              Cis:= Cis / two2power[ 29];
722
723              io:= Twoscomplement( Scale( tempephem, 9, 0, 32), 32);
724              io:= io / two2power[ 31];
725
726              Crc:= Twoscomplement( Scale( tempephem, 13, 0, 16), 16);
727              Crc:= Crc / two2power[ 5];
728
729              omega:= Twoscomplement( Scale( tempephem, 15, 0, 32),
730                                     32);
731              omega:= omega / two2power[ 31];
732
733              omegadot:= Twoscomplement( Scale( tempephem, 19, 0, 24),
734                                         24);
735              omegadot:= omegadot / two2power[ 43];
736
737              IDOT:= Twoscomplement( Scale( tempephem, 23, 0, 14), 14);
738              IDOT:= IDOT / two2power[ 43];
739
740              Date_and_Time( GPSint.prn[ sv_id].Tephem);
741              sv_id:= 0; { When last record for
742                        { satellite nr: sv_id is
743                        { used, then reset sv_id
744                        { so, no errors are made
745                        { if records 201..203 are
746                        { received without record
747                        { 200( correct sv_id)}
748
749              End; { End of Case 203}
750              End; { End of case}
751              End; { End of with}
752              End; { End of procedure Ephemeris}
753
754          {----- end included procedures collectport1-----}
755          Var
756              rec,
757              dumstr      :      String;
758              dumch       :      Char;
759              recnum,
760              code        :      Integer;
761              dum         :      Integer;
762              x           :      Byte;
763              time,
764              result      :      timetype;
765              part        :      String;
766
767          {----- start procedure collect port1-----}
768          Begin
769              GPSint:= tempGPSint; { update GPSint with temp}
770              If (Charsinbuff( port1) > 80) { orarily stored GPSint}
771              { only start if there}

```

```

771 Then Begin
772     rec:= Getcharbuff( port1);
773     { are enough chars in buff}
774     { Get first char}
775     If ( rec = LF)
776     { If this is a line feed}
777     Then Begin
778     { then we expect a new}
779     { leave a LF in buffer}
780     Repeat
781     { line to start}
782     dumch:= LookBuff( port1);
783     If ( dumch <> LF)
784     Then rec:= rec + Getcharbuff( port1);
785     Until ( dumch = LF) Or ( charsinbuff( port1)= 0);
786     { continue until end of
787     { line is detected, or
788     { no more chars}
789     { if no more chars, then
790     { error in received line}
791     If ( charsinbuff( port1) = 0)
792     Then Exit;
793     If ( rec[4] = ' ')
794     { for right alignment of
795     { record number}
796     Then rec:= ' ' + rec;
797     { insert space if necessary}
798     { copy the number}
799     dumstr:= Copy( rec, 1, 4);
800     { convert string to value}
801     Val( dumstr, recnum, code);
802     If ( code = 0)
803     { if no error converting}
804     Then Begin
805     gotoxy( 1, 17);
806     write( rec);
807     SaveEquipmentMessage( 'RG: ' + rec);
808     Case recnum Of
809     1 : Rawdata( rec, GPSint, svcount);
810     135 : Ionoscor( rec, GPSint);
811     200 : Get_SV_id( rec, sv_id);
812     201 : Clockinfo( rec, sv_id, GPSint);
813     202..203 : Ephemeris( recnum, rec, sv_id,
814     GPSint);
815     End;
816     { end case}
817     End
818     { end begin}
819     Else ;
820     { what to do if error?}
821     End
822     Else Begin
823     { if no begin of line}
824     Repeat
825     { skip chars from buffer}
826     dumch:= LookBuff( port1);
827     If ( dumch <> LF)
828     Then dumch:= Getcharbuff( port1);
829     Until ( dumch = LF) Or ( charsinbuff( port1) = 0);
830     { a lf was found or the
831     { buffer is empty}
832     End;
833     End; { end if charsinbuff}
834     tempGPSint:= GPSint;
835     Date_and_Time( time);

```

```

826 Addtime( Tpr, Valid_Tpr, result);
827 If Later( time, result) Or ( svcount = GPSint.numofsat)
828 Then Begin
829     For x:= 1 To 32 Do
830     tempGPSint.prn[x].flag:= True; { reset flags if all PR's
831     { are present}
832     svcount:= 0;
833     End
834     Else If (completeinfo = 1)
835     { If only complete info}
836     Then For x:= 1 To 32 Do
837     { should be transferred}
838     GPSint.prn[x].flag:= True; { keep flags True}
839     End; { end collectport1}
840     {----- end collectport1-----}
841     {----- end included procedures CollectGPSrec-----}
842 Var
843     counter : Integer;
844     {----- start procedure CollectGPSrec-----}
845 Begin
846     counter:= 1;
847     { now we read 2 message}
848     Repeat
849     { from the engine receiver}
850     Collectport0( GPSint);
851     Collectport1( GPSint);
852     { these contain most certain}
853     Dec( counter);
854     { ly all the measurements}
855     Until (counter = 0);
856 End;
857 {----- end procedure CollectGPSrec-----}
858 {----- start procedure execgpsrecommand-----}
859 Procedure ExecGPSrecCommand( command: commandtype);
860 { IN THE COMMENT, UNDER-
861 { SCORES ' ' SHOULD BE
862 { READ AS SPACES ' '}
863 Var
864     substr,
865     sendline : String;
866     code,
867     sv : Integer;
868     rate : Integer;
869 Begin
870     sendline:= '';
871     substr:= Copy( command, 5, Length( command) - 4);
872     { delete header: 'GPS:'}
873     If substr = 'RESET'
874     Then Begin
875     sendline:= '$PMVXG,018,T'#13#10; { RESTART THE ENGINE WITH
876     { TEPID START}
877     IWriteCom( port0, sendline);
878     SaveEquipmentMessage( 'SG: ' + sendline);
879     End;
880     { clear port0 outputlist}

```

```

881 If (Copy( substr, 1, 4) = 'INIT') And
882   ( Length( substr) >= 52)
883 Then Begin
884   sendline:= '$PMVXG,000,'+
885     Copy( substr, 6, 2) + ',' +
886     Copy( substr, 9, 2) + ',' + { INITIALISES THE ENGINE}
887     '19' +
888     Copy( substr, 12, 2) + ',' + { RECEIVER WITH POSITION}
889     Copy( substr, 15, 6) + ',' + { ETC. FORMAT IS: 'INIT_' }
890     Copy( substr, 22, 9) + ',' + { DD_MM_YY_HHMM_DDM.MMMM}
891     Copy( substr, 32, 1) + ',' + { _N_DDDMM.MMMM_E_HHHH.H_}
892     { AA.A_EL'}
893     Copy( substr, 34, 10)+ ',' + { WHERE DD_MM_YY IS THE}
894     Copy( substr, 45, 1) + ',' + { DATE IN DAY, MONTH, AND}
895     Copy( substr, 47, 7) + ',' +
896     '#13#10;
897
898   IWriteCom( port0, sendline);
899 SaveEquipmentMessage( 'SG: ' + sendline);
900
901   sendline:=
902     '$PMVXG,001,,, ' + { YEAR. HHMM IS TIME, }
903     Copy( substr, 55, 4) + ',' + { DDM.MMMM IS LATITUDE}
904     ',,, ' + { IN DEGREES AND MINUTES,}
905     Copy( substr, 60, 2) + ',' + { N IS NORTH OR SOUTH }
906     ' ,'+#13#10; { (N/S), DDDMM.MMMM IS }
907                   { LONGITUDE IN DEGREES }
908                   { AND MINUTES, E IS EAST}
909                   { OR WEST (E/W) AND }
910                   { HHHH.H IS ALTITUDE }
911                   { ABOVE MEAN SEA LEVEL,}
912                   { AA.A IS HORIZONTAL}
913                   { ACCELERATION FACTOR}
914                   { EL IS ELEVATION LIMIT}
915
916   IWriteCom( port0, sendline);
917 SaveEquipmentMessage( 'SG: ' + sendline);
918
919   sendline:= '$PMVXG,024,-,+,-,-,-,-'#13#10;
920   IWriteCom( port0, sendline);
921 SaveEquipmentMessage( 'SG: ' + sendline);
922
923   { NO NAV RESULT, RAW
924   MEASUREMENTS, APMANAC &
925   EPHEMERIS, NO CONTROL
926   INFO, NO TIME RECOVERY,
927   NO FULL DEBUG, NO PARTIAL
928   DEBUG}
929   sendline:= '$PMVXG,023,D,G,,,,,'#13#10;
930   IWriteCom( port0, sendline);
931 SaveEquipmentMessage( 'SG: ' + sendline);
932
933   { SET TIME RECOVERY ON}
934   sendline:= '$PMVXG,007,,1,,,,,'#13#10;
935   IWriteCom( port0, sendline); { NO OUTPUT ON CTRL PORT}
936   SaveEquipmentMessage( 'SG: ' + sendline);

```

```

936   sendline:= '$PMVXG,007,000,0,1,,1,,,,,'#13#10;
937   IWriteCom( port0, sendline); { OUTPUT MESSAGE 000}
938 SaveEquipmentMessage( 'SG: ' + sendline);
939   sendline:= '$PMVXG,007,001,0,1,,1,,,,,'#13#10;
940   IWriteCom( port0, sendline); { OUTPUT MESSAGE 001}
941 SaveEquipmentMessage( 'SG: ' + sendline);
942   sendline:= '$PMVXG,007,021,0,1,,1,,,,,'#13#10;
943   IWriteCom( port0, sendline); { OUTPUT MESSAGE 021}
944 SaveEquipmentMessage( 'SG: ' + sendline);
945
946   End;
947   If substr = 'SEND EPHEMERIS ETC'
948   Then Begin
949     sendline:= '$PMVXG,027,,,,2,2'#13#10;
950     IWriteCom( port0, sendline);
951 SaveEquipmentMessage( 'SG: ' + sendline);
952     End; { OUTPUT EPHEMERIS AND
953           ALMANAC NOW}
954
955 End;
956 {----- End procedure execgpsrecommand-----}
957
958 {----- start procedure closegpsrec-----}
959 Procedure CloseGPSRec;
960
961 Var
962   setupfile : Text;
963   value : String;
964
965 Begin
966   Removeint( port0);
967   Removeint( port1);
968
969   OpenConfigWrite( setupfile, MIAScfgname);
970   Writeln( setupfile, 'GPSENGINE');
971
972   Str( port0, value);
973   Writeln( setupfile, #9'port0 = ', value, ';');
974
975   Str( port1, value);
976   Writeln( setupfile, #9'port1 = ', value, ';');
977
978   Str( completeinfo, value);
979   Writeln( setupfile, #9'completeinfo = ', value, ';');
980   CloseConfig( setupfile);
981 End;
982 {----- end procedure closegpsrec-----}
983
984 {----- start initialising -----}
985 Begin { initialising part}
986   { used to make a table}
987   { containing the power of 2}

```

```

991     two2power[ 0]:= 1;           { fill most left position}
992     For x:= 1 To 55 Do          { calculate 2^1 to 2^55}
993         two2power[ x]:= two2power[ x-1] * 2;
994
995     sv_id:=0;                   { set sv_id to 0, which is
996                                 invalid, so no mistake can
997                                 be made}
998     ErrorTime( Tsv_id);
999     ErrorTime( Valid_Tsv_id);
1000     Valid_Tsv_id.minute:= 5;    { the sv_id stays valid for 5}
1001                                 { minutes ( see record 200)}
1002     ErrorTime( Tpr);
1003     ErrorTime( Valid_Tpr);      { the pr are maximum valid for}
1004     Valid_Tpr.sec:= 1;          { 1 second, this is the update}
1005                                 { rate of the GPSengine rx}
1006     old_user_ms:= 0;
1007     svcoun:= 0;
1008
1009     With tempGPSint Do          { make an empty GPSint set}
1010     Begin
1011         flag:= True;
1012         ErrorTime( Tionos);
1013         For x:= 1 To 32 Do
1014             With prn[x] Do
1015             Begin
1016                 flag:= True;
1017                 ErrorTime( Tck);
1018                 ErrorTime( Tephem);
1019             End;
1020             numofsat:= 0;
1021         End;
1022     End;
1023     completeinfo:= 1;
1024 End.
1025 {----- end initialising -----}
1026 {----- end Unit GPSENGINE -----}

```



```

1 Unit MLSbendix;
2 {*****}
3 { This unit is meant to be used with a Bendix MLS-20A receiver. This
4 receiver is a quasi MLSarinc727 receiver. The mls data words are not
5 passed through by this receiver. The basic datawords are simulated by
6 assigning variables with static basic data values}
7 {*****}
8
9 {$N+,E+}
10
11 Interface
12
13
14 Uses MIASglob, MLScglob, crt;
15
16
17 Procedure InitMLSrec( Var error: Boolean);
18 {*****}
19 { Initialise the Arinc 727 receiver. If something went wrong, error := True.
20 Input : -
21 Output: error}
22 {*****}
23
24
25 Procedure CollectMLSrec( Var MLSint: MLSinttype);
26 {*****}
27 { Get Arinc 429 words; Skip the words that are not necessary; check the
28 necessary words on parity. ADW's are checked on CRC.
29 Input : -
30 Output: MLSint}
31 {*****}
32
33
34 Procedure ExecMLSrecCommand( command: commandtype);
35 {*****}
36 { Convert a MLScommand, to a command word for the ARinc 727 MLS receiver.
37 That is, a Arinc 429 word. Then send the word using the Arinc 429 tx
38 channel.
39 Input : command
40 Output: Arinc 429 word on 429 tx port}
41 {*****}
42
43
44 Procedure CloseMLSrec;
45 {*****}
46 { Make sure the ARINC 727 MLS receiver is back to normal. Restore interrupt-
47 vectors etc.
48 Input : -
49 Output: -}
50 {*****}
51
52
53 Implementation
54
55 Uses Ar429comm, Ar429, Miscell, User, ADW;

```

```

56
57 Type
58   ADW_prestype      =      Array[1..4] Of Boolean;
59   ByteArray         =      Array[1..10] Of Byte;
60
61 Const
62   mlsfreq_lab=      036;           { mlsfrequency label}
63   azimuth_lab =      151;         { azimuth function label}
64   elevat_lab =      152;         { elevation function label}
65   BackAz_lab =      240;         { backazimuth function label}
66
67 Var
68   ADW_A_pres,
69   ADW_B_pres,
70   ADW_C_pres        :      ADW_prestype;
71   ADW_A,
72   ADW_B,
73   ADW_C             :      ADWtype;
74   x                 :      Integer;
75   AngleBegin        :      Boolean;
76   tempMLSint        :      MLSinttype;
77   Tangle,
78   Valid_Tangle      :      timetype;
79   completeinfo      :      Byte;
80   irq               :      Byte;
81   cardaddress       :      Word;
82   badwline          :      String;
83   bytes             :      ByteArray;
84   bytecount         :      Byte;
85   wordready         :      Boolean;
86
87 {-----Start InitMLSrec-----}
88 Procedure InitMLSrec( Var error: Boolean);
89
90 Var
91   title,
92   line             :      String;
93   code            :      Integer;
94   varname,
95   value           :      String;
96   setupfile       :      Text;
97   selecttable     :      arraytype;
98
99 Begin
100   error:= True;
101   cardaddress:= $280;
102   irq:= 7;
103   badwline:= '';
104   wordready:= False;
105   bytecount:= 1;
106
107
108   OpenConfigRead( setupfile, MIAScfname);
109   Repeat
110
111       { find MLSBENDIX part of
112       config file}

```

```

111      Readln( setupfile, title);
112      Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'MLSBENDIX'));
113
114      If Not Eof( setupfile)           { if there is more in file}
115      Then Repeat
116          Readln( setupfile, line);      { get a line}
117          Convert( line, varname, value); { extract the variable name
118                                          and value}
119          If ( varname = 'COMPLETEINFO')
120          Then Val( value, completeinfo, code);
121
122          If ( varname = 'CARDADDRESS')
123          Then Val( value, cardaddress, code);
124
125          If ( varname = 'IRQ')
126          Then Val( value, irq, code);
127      Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
128          { repeat until end of file}
129      CloseConfig( setupfile);
130
131      Ar429comm.InitAr429;
132          { Initialise comm-port}
133          { for VHF datalink}
134      selecttable[ 0]:= mlsfreq_lab;      { Initialise ARINC card}
135      selecttable[ 1]:= azimuth_lab;      { by selecting the labels}
136      selecttable[ 2]:= elevat_lab;       { for triggering}
137      selecttable[ 3]:= backaz_lab;
138
139      Ar429.InitAr429( error, selecttable, 4);
140
141      wordready:= False;
142      bytcount:= 1;
143
144      InitKaartadres( cardaddress);
145      KiesIRQ( IRQ);
146      Install_ADW_int;
147      ProgTrigFunktie( 1, $0A);           { Select BDW 1}
148      ProgTrigFunktie( 2, $1F);           { Select BDW 2}
149      ProgTrigFunktie( 3, $55);
150      ProgTrigFunktie( 4, $11);
151      ProgTrigFunktie( 5, $1B);
152      ProgTrigFunktie( 6, $58);           { Select BDW 6}
153      ProgTrigFunktie( 7, $27);           { Select ADW A}
154  End;
155  {-----End InitMLSrec-----}
156
157  {-----Start CollectMLSrec-----}
158  Procedure CollectMLSrec( Var MLSint: MLSinttype);
159
160      {----- start included procedures CollectMLSrec-----}
161      Function ADW_present( ADW_pres: ADW_prestype): Boolean;
162      {*****}
163      { Check if all 4 arinc429 words, that form an adw, are present. A
164      true is output if all 4 are present, if not, a false is output.
165      Input : Array with present flags

```

```

166      Output: Boolean.)
167      {*****}
168
169      Begin
170          ADW_present:= ADW_pres[ 1] And ADW_pres[ 2] And
171                      ADW_pres[ 3] And ADW_pres[ 4];
172      End;
173
174      Function Hamming_Fail( ADW: ADWtype): Boolean;
175      {*****}
176      { This function checks the hammingcode in the specified ADW. If the
177      hammingcode was correct, then the output is true. See Annex 10 p 150B
178      Input : ADW
179      Output: Boolean}
180      { Author: Maarten Uit de Haag;
181      Revised: Marco Meijer}
182      {*****}
183      Var
184          x      : Integer;
185          succes : Boolean;
186          check   : Array[ 0..6] Of Integer;
187
188      Begin
189          succes:= True;
190
191          For x:= 0 To 4 Do
192          Begin
193              check[ x]:= ADW[13 + x] + ADW[14 + x] + ADW[15 + x] +
194                      ADW[16 + x] + ADW[17 + x] + ADW[18 + x] +
195                      ADW[20 + x] + ADW[22 + x] + ADW[24 + x] +
196                      ADW[25 + x] + ADW[28 + x] + ADW[29 + x] +
197                      ADW[31 + x] + ADW[32 + x] + ADW[33 + x] +
198                      ADW[35 + x] + ADW[36 + x] + ADW[38 + x] +
199                      ADW[41 + x] + ADW[44 + x] + ADW[45 + x] +
200                      ADW[46 + x] + ADW[50 + x] + ADW[52 + x] +
201                      ADW[53 + x] + ADW[54 + x] + ADW[55 + x] +
202                      ADW[58 + x] + ADW[60 + x] + ADW[64 + x] +
203                      ADW[65 + x];
204              check[ x]:= check[ x] Mod 2;
205              succes:= succes And ( check[ x] = ADW[ 70 + x]);
206          End;
207          check[ 5]:= ADW[13 ] + ADW[14 ] + ADW[15 ] + ADW[16 ] +
208                      ADW[17 ] + ADW[19 ] + ADW[21 ] + ADW[23 ] +
209                      ADW[24 ] + ADW[27 ] + ADW[28 ] + ADW[30 ] +
210                      ADW[31 ] + ADW[32 ] + ADW[34 ] + ADW[35 ] +
211                      ADW[37 ] + ADW[40 ] + ADW[43 ] + ADW[44 ] +
212                      ADW[45 ] + ADW[49 ] + ADW[51 ] + ADW[52 ] +
213                      ADW[53 ] + ADW[54 ] + ADW[57 ] + ADW[59 ] +
214                      ADW[63 ] + ADW[64 ] + ADW[69 ];
215          check[ 5]:= check[ 5] Mod 2;
216          succes:= succes And ( check[ 5] = ADW[ 75]);
217
218          check[ 6]:= 0;
219          For x:= 13 To 75 Do

```

```

221      check[ 6] := check[ 6] + ADW[ x];
222      check[ 6] := check[ 6] Mod 2;
223      succes := succes And ( check[ 6] = ADW[ 76]);
224
225      Hamming_Fail := Not Succes;
226 End;
227
228
229 Function Adress_Fail( ADW: ADWtype): Boolean;
230 {*****}
231 { This function checks the parity of the address in the specified ADW.
232   If the address was correct, then the output is False. See Annex 10 p 150
233   Input : ADW
234   Output: Boolean}
235 { Author: Maarten Uit de Haag;
236   Revised: Marco Meijer}
237 {*****}
238 Var
239   check      :      Array[ 1..2] Of Integer;
240
241 Begin
242   check[ 1] := ADW[ 13] + ADW[ 14] + ADW[ 15] + ADW[ 16] + ADW[ 17] +
243     ADW[ 18];
244   check[ 1] := check[ 1] Mod 2;
245   check[ 2] := ADW[ 14] + ADW[ 16] + ADW[ 18];
246   check[ 2] := check[ 2] Mod 2;
247
248   Adress_Fail := Not ((check[ 1] = ADW[ 19]) And
249     ( check[ 2] = ADW[ 20]));
250 End;
251
252
253
254 Function ADW_adress( ADW: ADWtype): Byte;
255 {*****}
256 { This function takes bits 13 to 18 of the ADW and translates these
257   bits to an adress. This is a normal binary code. See Annex 10 p 150
258   Input : ADW
259   Output: adress}
260 {*****}
261 Var
262   adress,
263   mult,
264   x      :      Integer;
265
266 Begin
267   adress := 0;
268   mult := 32;
269   For x := 0 To 5 Do
270     Begin
271       adress := adress + ADW[ 13 + x] * mult;
272       mult := mult Div 2;
273     End;
274   ADW_adress := adress;
275 End;

```

```

276
277
278 Function Conv_ADW( ADW: ADWtype; start: Byte; number: Byte): Word;
279 {*****}
280 { This function converts single bits to a number. 'Start' indicates
281   the start bit in the ADW. First bit is bitnumber 1. 'Number' indicates
282   the number of bits to be used for the number to be formed. The number
283   is output as a Word. See Annex 10 p 60CC; LSB first
284   Input : ADW
285           startbit
286           number of bits
287   Output: number}
288 {*****}
289 Var
290   value,
291   mult      :      Word;
292
293 Begin
294   value := 0;
295   mult := 1;
296   For x := 0 To number - 1 Do
297     Begin
298       value := value + ADW[ start + x] * mult;
299       mult := mult * 2;
300     End;
301   Conv_ADW := value;
302 End;
303
304 Procedure ADW_A_conv( Var Mlsint: mlsinttype; ADW_A: ADWtype);
305
306 Var
307   adress      :      Byte;
308
309 Begin
310   adress := ADW_Adress( ADW_A);
311   Case adress Of
312     1: Begin
313       Date_and_Time( Mlsint.AuxA1_time);
314       With Mlsint.auxa1 Do
315         Begin
316           If ADW_A[ 30] = 1      { MSB is sign bit}
317             { see Annex 10 p150}
318             Then AzOff := -1 * Conv_ADW( ADW_A, 21, 9)
319             Else AzOff := Conv_ADW( ADW_A, 21, 9);
320
321           Az2MLSdatDist := Conv_ADW( ADW_A, 31, 13);
322
323           If ADW_A[ 55] = 1
324             Then AzAlignRun := -1 * Conv_ADW( ADW_A, 44, 11)
325             Else AzAlignRun := Conv_ADW( ADW_A, 44, 11);
326           AzAlignRun := AzAlignRun * 0.01;
327
328           AzCoordSys := ADW_A[ 56];
329         End;
330       Mlsint.AuxA1_flag := False;

```

```

331      End;
332      2:Begin
333          Date_and_Time( Mlsint.AuxA2_time);
334          With Mlsint.auxa2 Do
335              Begin
336                  If ADW_A[ 30] = 1
337                      Then ElOff:= -1 * Conv_ADW( ADW_A, 21, 9)
338                      Else ElOff:= Conv_ADW( ADW_A, 21, 9);
339
340                  MLSdat2thres:= Conv_ADW( ADW_A, 31, 10);
341
342                  If ADW_A[ 47] = 1
343                      Then ElHeight:= -1 * Conv_ADW( ADW_A, 41, 6)
344                      Else ElHeight:= Conv_ADW( ADW_A, 41, 6);
345                  ElHeight:= ElHeight * 0.1;
346              End;
347          Mlsint.Auxa2_flag:= False;
348      End;
349      3:Begin
350          Date_and_Time( Mlsint.AuxA3_time);
351          With Mlsint.auxa3 Do
352              Begin
353                  If ADW_A[ 30] = 1
354                      Then DMEoff:= -1 * Conv_ADW( ADW_A, 21, 9)
355                      Else DMEoff:= Conv_ADW( ADW_A, 21, 9);
356
357                  If ADW_A[ 44] = 1
358                      Then DME2MLSdatdist:= -1 * Conv_ADW( ADW_A, 31, 13)
359                      Else DME2MLSdatdist:= Conv_ADW( ADW_A, 31, 13);
360              End;
361          Mlsint.Auxa3_flag:= False;
362      End;
363      4:Begin
364          Date_and_Time( Mlsint.AuxA4_time);
365          With Mlsint.auxa4 Do
366              Begin
367                  If ADW_A[ 30] = 1
368                      Then BAZoff:= -1 * Conv_ADW( ADW_A, 21, 9)
369                      Else BAZoff:= Conv_ADW( ADW_A, 21, 9);
370
371                  BAZ2MLSdatdist:= Conv_ADW( ADW_A, 31, 11);
372
373                  If ADW_A[ 53] = 1
374                      Then BAZAlignRun:= -1 * Conv_ADW( ADW_A, 42, 11)
375                      Else BAZAlignRun:= Conv_ADW( ADW_A, 42, 11);
376                  BAZAlignRun:= BAZAlignRun * 0.01;
377              End;
378          Mlsint.Auxa3_flag:= False;
379      End;
380      End;
381      End;
382
383
384      Procedure ADW_A_coll( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype;
385                          a_label: byte; Var ADW_A_pres: ADW_prestype;
386
387                          Var ADW_A: ADWtype);
388      {*****
389      { This procedure converts the A-ADW's to Pascal variables.
390      { See Annex 10 p 150A etc.
391          Input : Arinc 429 word
392                Arinc 429 label
393                ADW_A_present array of boolean
394          Output: Mlsint
395                A-ADW )
396      {*****
397      Var
398          address      :      Byte;
399
400      Begin
401          Case a_label Of
402              88 :Begin      {130}
403
404
405
406
407                  For x:= 13 To 28 Do
408                      ADW_A[ x]:= Ar429word[ x - 13 + 14];
409                  ADW_A_pres[ 1]:= True;
410
411              End;
412              89 :Begin      {131}
413
414
415
416
417                  If Not ADW_A_pres[1]
418                      Then Exit;
419
420                  { take relevant bits from
421                  { Arinc 429 words and put
422                  { them in an ADW. update an
423                  { array of booleans to indi-
424                  { cate if the complete ADW
425                  { is present}
426
427                  For x:= 29 To 44 Do
428                      ADW_A[ x]:= Ar429word[ x - 29 + 14];
429                  ADW_A_pres[ 2]:= True;
430
431              End;
432              90 :Begin      {132}
433
434
435
436
437                  If Not ADW_A_pres[2]
438                      Then Exit;
439
440                  { third part only valid, if}
441                  { second and first part}
442                  { present}
443
444                  For x:= 45 To 60 Do
445                      ADW_A[ x]:= Ar429word[ x - 45 + 14];
446                  ADW_A_pres[ 3]:= True;
447
448              End;
449              91 :Begin      {133}
450
451
452
453
454                  If Not ADW_A_pres[3]
455                      Then Exit;
456
457                  { fourth part only valid, if}
458                  { third, second and first }
459                  { part present}
460
461                  For x:= 61 To 76 Do
462                      ADW_A[ x]:= Ar429word[ x - 61 + 14];
463                  ADW_A_pres[ 4]:= True;
464
465              End;
466          End;
467      End;
468      If (ADW_present( ADW_A_pres) And Not Hamming_Fail( ADW_A) And
469          Not Adress_Fail( ADW_A))
470          { If the ADW is valid and
471          { the address is correct,
472          { begin decoding the info}
473      Then Begin

```

```

441      ADW_A_conv( MLSint, ADW_A);
442      For x:= 1 To 4 Do
443          ADW_A_pres[x]:= False;
444      End;
445      { End of If}
446      { End of procedure}
447
448 {-----}
449 { The following part should not necessarily be implemented, because ADW B
450   and C are not yet assigned.}
451 {-----}
452
453 Procedure ADW_B_conv( Var MLSint: mlsinttype; ADW_B: ADWtype);
454
455 Begin
456     Date_and_Time( MLSint.AuxB_time);
457     MLSint.AuxB:= ADW_B;
458 End;
459
460 Procedure ADW_B_coll( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype;
461                      a_label: byte; Var ADW_B_pres: ADW_prestype;
462                      Var ADW_B: ADWtype);
463 {*****}
464 { This procedure converts the B-ADW's to Pascal variables.
465   Input : Arinc 429 word
466           Arinc 429 label
467           ADW_B_present array of boolean
468   Output: Mlsint
469           B-ADW }
470 {*****}
471 Var
472     adress      :      Byte;
473 Begin
474     Case a_label Of
475         92 :Begin      {134}
476             For x:= 13 To 28 Do
477                 ADW_B[ x]:= Ar429word[ x - 13 + 14];
478                 ADW_B_pres[ 1]:= True;
479             End;
480         93 :Begin      {135}
481             If Not ADW_B_pres[1]
482                 Then Exit;
483
484             For x:= 29 To 44 Do
485                 ADW_B[ x]:= Ar429word[ x - 29 + 14];
486                 ADW_B_pres[ 2]:= True;
487             End;
488         94 :Begin      {136}
489             If Not ADW_B_pres[2]
490                 Then Exit;
491
492             For x:= 45 To 60 Do
493                 ADW_B[ x]:= Ar429word[ x - 45 + 14];
494                 ADW_B_pres[ 3]:= True;
495             End;

```

```

496
497     If Not ADW_B_pres[3]
498         Then Exit;
499
500     For x:= 61 To 76 Do
501         ADW_B[ x]:= Ar429word[ x - 61 + 14];
502         ADW_B_pres[ 4]:= True;
503     End;
504 End;
505 If (ADW_present( ADW_B_pres) And Not Hamming_Fail( ADW_B) And
506     Not Adress_Fail( ADW_B))
507 Then Begin
508     ADW_B_conv( MLSint, ADW_B);
509     For x:= 1 To 4 Do
510         ADW_B_pres[x]:= False;
511     End;
512 End;
513
514 Procedure ADW_C_conv( Var MLSint: mlsinttype; ADW_C: ADWtype);
515
516 Begin
517     Date_and_Time( MLSint.AuxC_time);
518     MLSint.AuxC:= ADW_C;
519 End;
520
521 Procedure ADW_C_coll( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype;
522                      a_label: byte; Var ADW_C_pres: ADW_prestype;
523                      Var ADW_C: ADWtype);
524 {*****}
525 { This procedure converts the C-ADW's to Pascal variables.
526   Input : Arinc 429 word
527           Arinc 429 label
528           ADW_C_present array of boolean
529   Output: Mlsint
530           C-ADW }
531 {*****}
532 Var
533     adress      :      Byte;
534 Begin
535     Case a_label Of
536         96 :Begin      {140}
537             For x:= 13 To 28 Do
538                 ADW_C[ x]:= Ar429word[ x - 13 + 14];
539                 ADW_C_pres[ 1]:= True;
540             End;
541         97 :Begin      {141}
542             If Not ADW_C_pres[1]
543                 Then Exit;
544
545             For x:= 29 To 44 Do
546                 ADW_C[ x]:= Ar429word[ x - 29 + 14];
547                 ADW_C_pres[ 2]:= True;

```

```

551      End;
552      98 :Begin      {142}
553          If Not ADW_C_pres[2]
554          Then Exit;
555
556          For x:= 45 To 60 Do
557              ADW_C[ x]:= Ar429word[ x - 45 + 14];
558          ADW_C_pres[ 3]:= True;
559      End;
560      99 :Begin      {143}
561          If Not ADW_C_pres[3]
562          Then Exit;
563
564          For x:= 61 To 76 Do
565              ADW_C[ x]:= Ar429word[ x - 61 + 14];
566          ADW_C_pres[ 4]:= True;
567      End;
568      End;
569      If (ADW_present( ADW_C_pres) And Not Hamming_Fail( ADW_C) And
570          Not Adress_Fail( ADW_C))      ( If the ADW is valid and
571                                          the address is correct,
572                                          begin decoding the info)
573      Then Begin
574          For x:= 1 To 4 Do
575              ADW_C_pres[x]:= False;
576          End;
577      End;
578 {-----end of part-----}
579
580      Function Conv_BAS( ar429word: ar429wordtype; start: Byte; number: Byte)
581          : Word;
582      {*****}
583      { This function converts single bits to a number. 'Start' indicates
584        the start bit in the Arinc 429 word. 'Number' indicates the number of
585        bits to be used for the number to be formed. The number is output as a
586        Word. See Annex 10 p 60CC: LSB first.
587          Input : Arinc429 word
588                  startbit
589                  number of bits
590          Output: number}
591      {*****}
592      Var
593          value,
594          mult      :      Word;
595          x          :      Byte;
596
597      Begin
598          value:= 0;
599          mult:= 1;
600          For x:= 0 To number - 1 Do
601              Begin
602                  value:= value + ar429word[ start + x] * mult;
603                  mult:= mult * 2;
604              End;
605          Conv_BAS:= value;

```

```

606      End;
607
608
609
610      Procedure Bas_1_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype);
611      {*****}
612      { This procedure converts the Basic dataword nr 1 to Pascal variables.
613        See Annex 10 p 146.
614          Input : Arinc 429 word
615          Output: Mlsint}
616      {*****}
617      Begin
618          Date_and_Time( Mlsint.Bas1_time);
619
620          With Mlsint.Bas1 Do
621              Begin
622                  Az2Thresdist:= Conv_bas( Ar429word, 12, 6) * 100;
623                  AzPropCovNegLim:= Conv_bas( Ar429word, 18, 5) * 2;
624                  AzPropCovPosLim:= Conv_bas( Ar429word, 23, 5) * 2;
625                  Cleartype:= Ar429word[ 29];
626              End;
627          Mlsint. Bas1_flag:= False;
628      End;
629
630
631      Procedure Bas_2_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype);
632      {*****}
633      { This procedure converts the Basic dataword nr 2 to Pascal variables.
634        See Annex 10 p 146.
635          Input : Arinc 429 word
636          Output: Mlsint}
637      {*****}
638      Begin
639          Date_and_Time( Mlsint.Bas2_time);
640
641          With Mlsint.Bas2 Do
642              Begin
643                  MinGP:= Conv_bas( Ar429word, 12, 7) * 0.1;
644                  BAZstat:= Ar429word[ 19];
645                  DMEstat:= Conv_bas( ar429word, 20, 2);
646                  Azstat:= Ar429word[ 22];
647                  Elstat:= Ar429word[ 23];
648              End;
649          Mlsint. Bas2_flag:= False;
650      End;
651
652
653      Procedure Bas_3_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype);
654      {*****}
655      { This procedure converts the Basic dataword nr 3 to Pascal variables.
656        See Annex 10 p 147.
657          Input : Arinc 429 word
658          Output: Mlsint}
659      {*****}
660      Begin

```

```

661     Date_and_Time( Mlsint.Bas3_time);
662
663     With Mlsint.Bas3 Do
664     Begin
665         AzBW:= Conv_bas( Ar429word, 12, 3) * 0.5;
666         ElBW:= Conv_bas( Ar429word, 15, 3) * 0.5;
667         DMEdist:= Conv_bas( Ar429word, 18, 9) * 12.5;
668     End;
669     Mlsint. Bas3_flag:= False;
670 End;
671
672
673 Procedure Bas_4_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
674 {*****}
675 { This procedure converts the Basic dataword nr 4 to Pascal variables.
676   See Annex 10 p 147.
677       Input : Arinc 429 word
678       Output: Mlsint}
679 {*****}
680 Begin
681     Date_and_Time( Mlsint.Bas4_time);
682
683     With Mlsint.Bas4 Do
684     Begin
685         AzMagOr:= Conv_bas( Ar429word, 12, 9);
686         BazMagor:= Conv_bas( Ar429word, 21, 9);
687     End;
688     Mlsint. Bas4_flag:= False;
689 End;
690
691
692 Procedure Bas_5_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
693 {*****}
694 { This procedure converts the Basic dataword nr 5 to Pascal variables.
695   See Annex 10 p 147.
696       Input : Arinc 429 word
697       Output: Mlsint}
698 {*****}
699 Begin
700     Date_and_Time( Mlsint.Bas5_time);
701
702     With Mlsint.Bas5 Do
703     Begin
704         BAZPropCovNegLim:= Conv_bas( Ar429word, 12, 5) * 2;
705         BazPropCovPosLim:= Conv_bas( Ar429word, 17, 5) * 2;
706         BazBW:= Conv_bas( Ar429word, 22, 3) * 0.5;
707         Bazstat:= Ar429word[ 25];
708     End;
709     Mlsint. Bas5_flag:= False;
710 End;
711
712
713 Procedure Bas_6_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
714 {*****}
715 { This procedure converts the Basic dataword nr 6 to Pascal variables.

```

```

716     See Annex 10 p 147.
717
718       Input : Arinc 429 word
719       Output: Mlsint}
720 {*****}
721 Begin
722     Date_and_Time( Mlsint.Bas6_time);
723
724     With Mlsint.Bas6 Do
725     Begin
726         MLSident[1]:= Char( Conv_bas( ar429word, 12, 6));
727         MLSident[2]:= Char( Conv_bas( ar429word, 18, 6));
728         MLSident[3]:= Char( Conv_bas( ar429word, 24, 6));
729     End;
730     Mlsint. Bas6_flag:= False;
731 End;
732
733 Procedure EL_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
734 {*****}
735 { This procedure converts the Arinc 429 word containing the glidepath
736   information to a Pascal variable.
737       Input : Arinc 429 word
738       Output: Mlsint}
739 {*****}
740 Begin
741     With Mlsint Do
742     Begin
743         If ( Ar429word[ 27] = 1) And
744             ( Ar429word[ 28] = 1) And
745             ( Ar429word[ 29] = 1)
746         Then Begin
747             ELangle:= Conv_bas( Ar429word, 13, 14);
748             ELangle:= ELangle - $4000;
749             ELangle:= ELangle * 0.005;
750         End;
751
752         If ( Ar429word[ 27] = 0) And
753             ( Ar429word[ 28] = 0) And
754             ( Ar429word[ 29] = 0)
755         Then ELangle:= Conv_bas( Ar429word, 13, 14) * 0.005;
756
757         ELantInUse:= Ar429word[ 12] + 1;      { bendix: 0 = aft ant}
758                                             {          1 = forward ant}
759                                             { ARinc: 1,2,3 = ant no}
760
761         ELangle_flag:= False;
762
763         If (Not anglebegin) Or (Not ELangle_flag)
764         Then Date_and_time( Tangle);
765     End;
766 End;
767
768 Procedure AZ_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
769 {*****}
770 { This procedure converts the Arinc 429 word containing the azimuthangle

```

```

771 information to a Pascal variable.
772         Input : Arinc 429 word
773         Output: Mlsint)
774 {*****}
775 Var
776     x,
777     cnt      :      Byte;
778
779 Begin
780     With Mlsint Do
781     Begin
782         If ( Ar429word[ 27] = 1) And
783             ( Ar429word[ 28] = 1) And
784             ( Ar429word[ 29] = 1)
785         Then Begin
786             Azangle:= Conv_bas( Ar429word, 13, 14);
787             Azangle:= Azangle - $4000;
788             Azangle:= Azangle * 0.005;
789         End;
790
791         If ( Ar429word[ 27] = 0) And
792             ( Ar429word[ 28] = 0) And
793             ( Ar429word[ 29] = 0)
794         Then Azangle:= Conv_bas( Ar429word, 13, 14) * 0.005;
795
796         AzAntInUse:= Ar429word[ 12] + 1;      { bendix: 0 = aft ant}
797                                         {      1 = forward ant}
798                                         { ARinc: 1,2,3 = ant no}
799
800         cnt:= 0;                          { extra check possible :}
801         For x:= 13 to 28 Do                { |angle| < propcoverage}
802             cnt:= cnt + Ar429word[ x];
803
804         If ( cnt = 0) And ( AR429word[ 29] = 1)
805         Then Leftclr:= True
806         Else Leftclr:= False;
807
808         If ( cnt = 16) And ( Ar429word[ 29] = 0)
809         Then Rightclr:= True
810         Else Rightclr:= False;
811
812         If Leftclr Or Rightclr
813         Then Azangle_flag:= True
814         Else Begin
815             AZangle_flag:= False;
816
817             If (Not anglebegin) Or (Not Azangle_flag)
818             Then Date_and_time( Tangle);
819         End;
820     End;
821 End;
822
823
824 Procedure BAZ_conv( Var Mlsint: Mlsinttype; Ar429word: Ar429wordtype);
825 {*****}

```

```

826 { This procedure converts the Arinc 429 word containing the backazimuth
827 angle information to a Pascal variable.
828         Input : Arinc 429 word
829         Output: Mlsint)
830 {*****}
831 Var
832     cnt,
833     x      :      Byte;
834
835 Begin
836     With Mlsint Do
837     Begin
838         If ( Ar429word[ 27] = 1) And
839             ( Ar429word[ 28] = 1) And
840             ( Ar429word[ 29] = 1)
841         Then Begin
842             BAZangle:= Conv_bas( Ar429word, 13, 14);
843             BAZangle:= BAZangle - $4000;
844             BAZangle:= BAZangle * 0.005;
845         End;
846
847         If ( Ar429word[ 27] = 0) And
848             ( Ar429word[ 28] = 0) And
849             ( Ar429word[ 29] = 0)
850         Then BAZangle:= Conv_bas( Ar429word, 13, 14) * 0.005;
851
852         AzAntInUse:= Ar429word[ 12] + 1;      { bendix: 0 = aft ant}
853                                         {      1 = forward ant}
854                                         { ARinc: 1,2,3 = ant no}
855
856         cnt:= 0;
857         For x:= 13 to 28 Do
858             cnt:= cnt + Ar429word[ x];
859
860         If ( cnt = 0) And ( AR429word[ 29] = 1)
861         Then Leftclr:= True
862         Else Leftclr:= False;
863
864         If ( cnt = 16) And ( Ar429word[ 29] = 0)
865         Then Rightclr:= True
866         Else Rightclr:= False;
867
868         If Leftclr Or Rightclr
869         Then BAZangle_flag:= True
870         Else Begin
871             BAZangle_flag:= False;
872
873             If (Not anglebegin) Or (Not BAZangle_flag)
874             Then Date_and_time( Tangle);
875         End;
876     End;
877 End;
878
879 Procedure Discretes_conv( Var Mlsint: Mlsinttype;
880                          Ar429word: Ar429wordtype);

```



```

881 {*****}
882 { This procedure converts the Arinc 429 word containing the MLS discretes
883   to Pascal variables.
884       Input : Arinc 429 word
885       Output: Mlsint}
886 {*****}
887 Begin
888     With Mlsint.discretes Do
889     Begin
890         antenna:= Conv_bas( Ar429word, 11, 2);
891         test      := Ar429word[ 13];
892         Azsource   := Ar429word[ 14];
893         Azselwarn  := Ar429word[ 15];
894         Bazselwarn := Ar429word[ 16];
895         GPselwarn  := Ar429word[ 17];
896         BAZavail   := Ar429word[ 18];
897         BAZeven    := Ar429word[ 19];
898         tuningcom  := Ar429word[ 20];
899         nr1antsel  := Ar429word[ 21];
900         changeinh  := Ar429word[ 22];
901         tunprtsel  := Ar429word[ 23];
902     End;
903     Mlsint. discretes_flag:=False;
904 End;
905
906 Function Hex2String( x: byte): string;
907 {*****}
908 { This function converts a byte to its hexadecimal representation in a
909   string. Written by Rob Luxen.
910       Input : Byte;
911       Output: A string;
912 {*****}
913 Const
914     hex: Array[ 0..15] Of Char = '0123456789ABCDEF';
915
916 Var
917     i :      Byte;
918
919 Begin
920     Hex2String[ 0]:= Chr(2);
921     For i:= 0 to 1 Do
922     Begin
923         hex2String[2-i]:= Hex[ x and $000F];
924         x:= x shr 4;
925     End;
926 End;
927 {-----end included procedures CollectMLSrec-----}
928
929 Var
930     line,
931     part      :   String;
932     Ar429word :   Ar429wordtype;
933     a_label   :   Byte;
934     adress    :   Byte;
935     NoWord    :   Boolean;

```

```

936     time,
937     result    :   timetype;
938     d,d1,i,j  :   Byte;
939     ADword    :   ADWtype;
940
941 {-----Start procedure CollectMLSrec-----}
942 Begin      { begin procedure CollectMLSrec}
943     Mlsint:= tempMLSint;
944
945     Ar429comm.GetAr429word( Ar429word, NoWord, a_label);
946                                     { the label is in dec}
947     Gotoxy( 1, 13);
948     If Not ( Noword)
949     Then Begin
950         Str( a_label: 3, part);
951         line:= part + ' ';
952         For x:= 1 To 32 Do
953         Begin
954             Str( ar429word[x]:1, part);
955             line:= line + part;
956         End;
957         Write( line, ' ');
958
959         SaveEquipmentMessage( 'RM: ' + line);
960     End;
961
962     Case a_label Of
963         88..91: {130..133} ADW_A_coll( Mlsint, Ar429word, a_label,
964                                         ADW_A_pres, ADW_A);
965         92..95: {134..137} ADW_B_coll( Mlsint, Ar429word, a_label,
966                                         ADW_B_pres, ADW_B);
967         96..99: {140..143} ADW_C_coll( Mlsint, Ar429word, a_label,
968                                         ADW_C_pres, ADW_C);
969         110 : {156}      Bas_1_conv( Mlsint, Ar429word);
970         111 : {157}      Bas_2_conv( Mlsint, Ar429word);
971         112 : {160}      Bas_3_conv( Mlsint, Ar429word);
972         113 : {161}      Bas_4_conv( Mlsint, Ar429word);
973         114 : {162}      Bas_5_conv( Mlsint, Ar429word);
974         115 : {163}      Bas_6_conv( Mlsint, Ar429word);
975         106 : {152}      EL_conv( Mlsint, Ar429word);
976         105 : {151}      Az_conv( Mlsint, Ar429word);
977         126 : {176}      Baz_conv( Mlsint, Ar429word);
978         184 : {270}      Discretes_conv( Mlsint, Ar429word);
979     End;
980
981     Ar429.GetAr429word( Ar429word, NoWord, a_label);
982                                     { the label is in dec}
983     If Not ( Noword) Then Begin
984         Str( a_label: 3, part);
985         line:= part + ' ';
986         For x:= 1 To 32 Do
987         Begin
988             Str( ar429word[x]:1, part);
989             line:= line + part;
990         End;
991         WriteLn( line);

```

```

991                                     1046
992 SaveEquipmentMessage( 'RM: ' + line); 1047
993 End;                                     1048
994                                     1049
995         Case a_label Of                 1050
996         88..91: (130..133) ADW_A_coll( 1051
997                                     ADW_A_pres, ADW_A); 1052
998         92..95: (134..137) ADW_B_coll( 1053
999                                     ADW_B_pres, ADW_B); 1054
1000        96..99: (140..143) ADW_C_coll( 1055
1001                                     ADW_C_pres, ADW_C); 1056
1002        110 : (156) Bas_1_conv( MLSint, Ar429word); 1057
1003        111 : (157) Bas_2_conv( MLSint, Ar429word); 1058
1004        112 : (160) Bas_3_conv( MLSint, Ar429word); 1059
1005        113 : (161) Bas_4_conv( MLSint, Ar429word); 1060
1006        114 : (162) Bas_5_conv( MLSint, Ar429word); 1061
1007        115 : (163) Bas_6_conv( MLSint, Ar429word); 1062
1008        106 : (152) EL_conv( MLSint, Ar429word); 1063
1009        105 : (151) Az_conv( MLSint, Ar429word); 1064
1010        126 : (176) Baz_conv( MLSint, Ar429word); 1065
1011        184 : (270) Discretes_conv( MLSint, Ar429word); 1066
1012        End;                             1067
1013                                     1068
1014        tempMLSint:= MLSint;              1069
1015                                     1070
1016        With Fifo Do                     1071
1017        Begin                             1072
1018            If Not FifoEmpty              1073
1019            Then Begin                    1074
1020                d:= GetFifo;               1075
1021                If d = $AA                 1076
1022                Then Begin                1077
1023                    If Not FifoEmpty      1078
1024                    Then Begin            1079
1025                        d1:= GetFifo;      1080
1026                        If d1 = $55        1081
1027                        Then Begin         1082
1028                            wordready:= True; 1083
1029                            SaveEquipmentMessage( 'RM: ' + badwline); 1084
1030                            badwline:= ''; 1085
1031                            End;           1086
1032                            Else Begin    1087
1033                                bytes[ bytecount]:= d; 1088
1034                                Inc( bytecount); 1089
1035                                bytes[ bytecount]:= d1; 1090
1036                                Inc( bytecount); 1091
1037                                wordready:= False; 1092
1038                                badwline:= badwline + 1093
1039                                    Hex2String( d) + 1094
1040                                    ' '; 1095
1041                                badwline:= badwline + 1096
1042                                    Hex2String( d1) + 1097
1043                                    ' '; 1098
1044                                End; 1099
1045                                End; 1100

```

```

1101
1102 With MLSint Do           { This part is to simulate}
1103 Begin                   { basic datawords.}
1104                       { In the bendix receiver,}
1105                       { they are not available}
1106     bas1.Az2thresdist    := Az2thresdist;
1107     bas1.AzPropCovNegLim := AzPropCovNegLim;
1108     bas1.AzPropCovPosLim := AzPropCovPosLim;
1109     bas1.Cleartype       := Cleartype;
1110
1111     Date_and_time( bas1_time);
1112     bas2.MinGP           := MinGP;
1113     bas2.BAZstat         := Bazstat;
1114     bas2.DMEstat         := DMEstat;
1115     bas2.Azstat          := Azstat;
1116     bas2.Elstat          := Elstat;
1117
1118     Date_and_time( bas2_time);
1119
1120     bas3.AzBW            := AzBW;
1121     bas3.ElBW            := ElBW;
1122     bas3.DMEdist         := DMEdist;
1123
1124     Date_and_time( bas3_time);
1125
1126     bas4.AzMagOr         := AzMagOr;
1127     bas4.BazMagOr        := BazMagOr;
1128
1129     Date_and_time( bas4_time);
1130
1131     bas5.BazPropCovNegLim:= BazPropCovNegLim;
1132     bas5.BazPropCovPosLim:= BazPropCovPosLim;
1133     bas5.BazBW           := BazBW;
1134     bas5.BazStat         := BazStat;
1135
1136     Date_and_time( bas5_time);
1137
1138     bas6.MLSident        := MLSident;
1139
1140     Date_and_time( bas6_time);
1141
1142     auxa1.AzOff           := AzOff;
1143     auxa1.Az2MLSdatdist  := Az2MLSdatdist;
1144     auxa1.AzAlignRun     := AzAlignRun;
1145     auxa1.AzCoorSyst     := AzCoorSyst;
1146
1147     Date_and_time( AuxA1_time);
1148
1149     auxa2.ElOff           := ElOff;
1150     auxa2.MLSdat2thres    := MLSdat2thres;
1151     auxa2.ElHeight        := ElHeight;
1152
1153     Date_and_time( AuxA2_time);
1154
1155     auxa3.DMEoff          := DMEoff;
1156
1157     auxa3.DME2MLSdatDist := DME2MLSdatDist;
1158
1159     Date_and_time( AuxA3_time);
1160
1161     auxa4.BAZOff          := BazOff;
1162     auxa4.BAZ2MLSdatDist := Baz2MLSdatdist;
1163     auxa4.BAZAlignRun     := BazAlignRun;
1164
1165     Date_and_time( AuxA4_time);
1166 End;
1167
1168 Date_and_Time( time);
1169
1170 With MLSint Do
1171 Begin
1172     Addtime( bas2_time, Valid_bas2, result);
1173     If Not Later( time, result) { determine if all info
1174     Then Begin                  { for a cycle is present}
1175         If (( bas2.Azstat = 1) And ( bas2.Elstat = 1) And
1176         ( Azangle_flag XOR Elangle_flag))
1177         Then anglebegin:= True
1178         Else anglebegin:= False;
1179     End;
1180
1181     Addtime( Tangle, Valid_Tangle, result);
1182     If Later( time, result) Or ( anglebegin = False)
1183     Then With tempMLSint Do { when the info is passed}
1184     Begin                   { through, make flags in}
1185         AZangle_flag:= True; { temporarily variable}
1186         Elangle_flag:= True; { True, to prevent sending}
1187         Bazangle_flag:= True; { info twice}
1188         DME_flag:= True;
1189         RightClr:= True;
1190         LeftClr:= True;
1191         Discretes_Flag:= True;
1192     End
1193     Else If ( completeinfo = 1)
1194     Then With MLSint Do
1195     Begin
1196         AZangle_flag:= True;
1197         Elangle_flag:= True;
1198         Bazangle_flag:= True;
1199         DME_flag:= True;
1200         RightClr:= True;
1201         LeftClr:= True;
1202         Discretes_Flag:= True;
1203     End;
1204 End;
1205 {-----End procedure CollectMLSrec-----}
1206
1207
1208
1209
1210 {-----Start procedure Execmlsrecommand-----}

```

```

1211 Procedure ExecMLSrecCommand( command: commandtype);
1212
1213 Const
1214     Prate      =      1000;
1215
1216 Var
1217     error      :      Boolean;
1218     datain     :      Longint;
1219     oct_lab    :      Byte;
1220     part,
1221     line       :      String;
1222     ar429word  :      ar429wordtype;
1223
1224 Begin
1225     { room for decoding the command. The result should be a valid
1226       arinc429 word of 32 bits. This should be coded in a 32 bit longint}
1227
1228     Ar429.SendAr429word( dataIn, oct_lab, Prate, error);
1229
1230 line:= '';
1231 For x:= 1 To 32 Do
1232 Begin
1233     Str( ar429word[x]:1, part);
1234     line:= line + part;
1235 End;
1236 SaveEquipmentMessage( 'SM: ' + line);
1237 End;
1238 {-----End procedure Execmlsrecommand-----}
1239
1240
1241 {-----Start procedure Closemlsrec-----}
1242 Procedure CloseMLSrec;
1243
1244 Var
1245     setupfile  :      Text;
1246     value      :      String;
1247
1248 Begin
1249     Ar429comm.CloseAR429;
1250     Ar429.CloseAr429;
1251     Remove_ADW_int;
1252
1253     OpenConfigWrite( setupfile, MIAScfname);
1254     Writeln( setupfile, 'MLSBENDIX');
1255
1256     Str( completeinfo, value);
1257     Writeln( setupfile, #9'completeinfo = ', value, '');
1258
1259     Str( cardaddress, value);
1260     Writeln( setupfile, #9'cardaddress = ', value, '');
1261
1262     Str( irq, value);
1263     Writeln( setupfile, #9'irq = ', value, '');
1264
1265     CloseConfig( setupfile);

```

```

1266 End;
1267 {-----End procedure Closemlsrec-----}
1268
1269
1270 {-----Start initialising-----}
1271 Begin
1272     For x:= 1 To 4 Do
1273     Begin
1274         ADW_A_pres[x]:= False;
1275         ADW_B_pres[x]:= False;
1276         ADW_C_pres[x]:= False;
1277     End;
1278     For x:= 13 To 76 Do
1279     Begin
1280         ADW_A[ x]:= 0;
1281         ADW_B[ x]:= 0;
1282         ADW_C[ x]:= 0;
1283     End;
1284
1285     With TempMLSint Do
1286     Begin
1287         Bas1_flag      := True;
1288         Bas2_flag      := true;
1289         Bas3_flag      := true;
1290         Bas4_flag      := true;
1291         Bas5_flag      := true;
1292         Bas6_flag      := true;
1293         auxa1_flag     := true;
1294         auxa2_flag     := true;
1295         auxa3_flag     := true;
1296         auxa4_flag     := true;
1297         ELangle_flag   := true;
1298         AZangle_flag   := true;
1299         BAZangle_flag  := true;
1300         DME_flag       := true;
1301         discretess_flag:= true;
1302         leftclr        := true;
1303         rightclr       := true;
1304         flag           := true;
1305
1306         ErrorTime( Bas1_time);
1307         ErrorTime( Bas2_time);
1308         ErrorTime( Bas3_time);
1309         ErrorTime( Bas4_time);
1310         ErrorTime( Bas5_time);
1311         ErrorTime( Bas6_time);
1312         ErrorTime( AuxA1_time);
1313         ErrorTime( AuxA2_time);
1314         ErrorTime( AuxA3_time);
1315         ErrorTime( AuxA4_time);
1316         ErrorTime( AuxB_time);
1317         ErrorTime( AuxC_time);
1318     End;
1319
1320     ErrorTime( Tangle);

```

Page 13, listing of MLSBENDI.PAS, date is 18-02-93, file date is 17-02-93, size is 50616 bytes.

```
1321      ErrorTime( Valid_Tangle);           { angles are no longer}
1322      Valid_Tangle.sec100:= 8;             { valid than 80 msec}
1323
1324      anglebegin:= False;
1325      completeinfo:= 1;
1326 End.
1327 {-----End initialising-----}
1328 {-----End Unit MLSBENDIX-----}
```